aktiv computern

Der BASIC-Interpreter im Laser 110, 210, 310 und VZ 200

Gerhard Wolf



Mein Home-Computer

Gerhard Wolf

Der BASIC-Interpreter im Laser 110, 210, 310 und VZ 200

HC – Mein Home-Computer

Gerhard Wolf

Der BASIC-Interpreter im Laser 110, 210, 310 und VZ 200

Aufbau und Arbeitsweise



VOGEL-BUCHVERLAG
WÜRZBURG

Vom selben Autor sind erschienen:
ROM-Listings für Laser 110, 210, 310 und VZ 200
(HC – Mein Home-Computer)
ISBN 3-8023–0852-2

Das Laser-DOS für Laser 110, 210, 310 und VZ 200
(HC – Mein Home-Computer)
ISBN 3-8023-0868-9

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Wolf, Gerhard:
Der BASIC-Interpreter im Laser 110, 210, 310 und VZ 200: Aufbau und Arbeitsweise / Gerhard Wolf. – 1. Aufl. – Würzburg: Vogel, 1985
(HC – Mein Home-Computer)
ISBN 3-8023-0874-3

ISBN 3-8023-0874-3
1. Auflage. 1985

Umschlaggestaltung: Bernd Schröder, Böhl
Herstellung: Alois Erdl KG, Trostberg

0. Einleitung

Die kleinen Computer der Serie LASER 110, 210, 310 und die VZ200 verdanken ihre Popularität nicht zuletzt dem komfortablen und umfangreichen BASIC-Interpreter, der sich in Speicherbausteinen (ROMs) im Innern des Rechners befindet und allen Nutzern nach dem Einschalten in all seinen Funktionen vollständig zur Verfügung steht.

Es handelt sich hier um eine leicht modifizierte Variante des bekannten und weltweit verbreiteten MICROSOFT BASIC, das Bestandteil einer Grund-Betriebssoftware ist. Diese Betriebssoftware besteht aus einem Grund-Betriebssystem und dem umfangreichen BASIC-Interpreter.

Als Erweiterung dazu dient ein Disketten-Betriebssystem (DOS = Disk Operating System). Dieses ist ebenfalls in ROM-Bausteinen fest gespeichert, ist jedoch nicht im Hauptrechner untergebracht, sondern befindet sich in der Diskettensteuerung. Durch Anschluß der Diskettensteuerung an den System-Bus werden diese ROM-Bausteine aktiviert und fügen die erforderliche Software zum Betrieb der Diskettenlaufwerke dem Grundsystem hinzu.

Eine weitere Leistungssteigerung läßt sich durch ein EXTENDED BASIC - Paket erreichen, das in Deutschland entwickelt wurde und den Sprachumfang des eingebauten BASIC-Interpreters um mehr als 30 Befehle erweitert, darunter so leistungsstarke wie PLOT, PAINT, CIRCLE, RENUMBER und viele andere mehr.

Ziel dieses Buches ist es, die wesentlichen Funktionen des BASIC-ROMs zu beschreiben, damit Sie die internen Vorgänge in Ihrem Rechner besser verstehen lernen und alle Funktionen optimal nutzen können. Das Buch soll auch dem Assembler-/Maschinenprogramm-Experten die Möglichkeiten eröffnen, Funktionen des BASIC-ROMs in eigenen Programmen zu nutzen, sei es, um einfache Daten-Konvertierungen auszuführen, die Ein-/Ausgabeschnittstellen zu benutzen oder mathematische Funktionen (z.B. Sinus, Kosinus usw.) nicht selber programmieren zu müssen.

Es kann nicht jede Routine bis ins letzte Detail beschrieben werden. Die hier enthaltene Beschreibung müßte jedoch ausreichen, darüber hinausgehende Detailuntersuchungen in der ROM-Auflistung selbst vornehmen zu können.

Eine detailliert dokumentierte ROM-Auflistung und eine ausführliche Beschreibung des Disketten-Betriebssystems für LASER 110-310 und VZ200 sind als eigenständige Bände im Vogel-Verlag erschienen.

Mein Dank gilt Herrn Dieter Effkemann für seine Unterstützung und Zuarbeit zum Kapitel 12 "Funktionsableitungen" und meinem Sohn Rainer für das Austesten der Programmbeispiele und mühevolles Korrekturlesen.

1. Was ist ein Betriebssystem

Ein Rechner ohne jede Art von Betriebssystem ist ein nutzloser Kasten, gefüllt mit elektronischen Bauteilen.

Der Bedarf nach einem Betriebssystem ergibt sich allein schon aus der Notwendigkeit, eine Kommunikationsmöglichkeit zwischen dem Rechner und seiner Umwelt herstellen zu müssen. Dazu gehört u.a. die ständige Überwachung aller Eingänge, z.B. der Tastatur, um Anweisungen und Daten entgegennehmen zu können und auch die Ausgabe von Daten, z.B. auf einen angeschlossenen Bildschirm, um Ergebnisse darstellen bzw. übermitteln zu können.

Wenn Sie an Ihrem LASER-Computer ein BASIC-Programm eintippen und anschließend ausführen lassen wollen, so muß es im Rechner bereits ein Programm geben, das Ihre Eingaben zunächst einmal entgegennimmt und an die richtige Stelle des Speichers bringt. Auch der Ablauf des Programms bedarf starker Unterstützung dieses internen Programms. Ein solches im Rechner vorhandenes Programm ist das Betriebssystem.

Es gibt mittlerweile Tausende verschiedener Betriebssysteme, vom einfachen System im ROM der Home- und Personal-Computer bis zu komplexen Gebilden, die bei Großrechenanlagen ganze Platteneinheiten belegen können.

Die Unterschiede sind zum einen durch die verschiedenen Rechner mit der unterschiedlichsten Hardwareausstattung bedingt, zum anderen aber auch durch die Anforderungen, die an ein solches System gestellt werden. Das Betriebssystem eines Rechners für eine Prozeßsteuerung sieht bestimmt ganz anders aus als das für eine kaufmännische Anwendung, obwohl beide ggf. auf der gleichen Hardware ablauffähig sind.

Aus diesem Grunde läßt sich auch keine genaue Definition eines Betriebssystems aufstellen.

Trotz aller Unterschiede sind jedoch die Grundbausteine der Betriebssysteme ähnlich und ganz allgemein lassen sich funktionell folgende Bausteine in den meisten Systemen erkennen:

1. Ein Monitor-Programm, das ständig alle Systemeingänge überwacht (z.B. die Tastaturabfrage).

2. Geräte-Treiberroutinen, die die speziellen physikalischen Bedürfnisse der angeschlossenen peripheren Geräte wie Tastatur, Bildschirm, Kassette, Diskette oder Drucker befriedigen.

3. Allgemeine Dienstroutinen, die nach Eingabe bestimmter Kommandos Systemfunktionen initialisieren (im LASER z.B. LIST, CLOAD).

4. Sprachumsetzer (Compiler, Assembler, Interpreter), die den Einsatz einer Programmiersprache ermöglichen (z.B. BASIC, PASCAL, FORTRAN).

5. Laufzeit-Unterstützungsroutinen für die jeweilige Programmiersprache. Dies sind u.a. die arithmetischen und mathematischen Funktionen, für die fertige Routinen im Betriebssystem vorhanden sind, die von der jeweiligen Sprache nur aufgerufen werden.

6. Hilfsroutinen zur Geräte- und Speicherverwaltung. Diese pflegen interne Tabellen, verwalten die verschiedenen Speicherbereiche und steuern den Zugriff zu den Ein-/Ausgabegeräten.

In den folgenden Abschnitten sollen diese Komponenten innerhalb des LASER-Betriebssystems identifiziert und beschrieben werden.

Da sich beim LASER-Computer die Betriebssoftware in ROM-Bausteinen innerhalb des Rechners befindet, ist sie nach Einschalten sofort in all ihren Funktionen verfügbar.

Bei anderen Rechnern und Betriebssystemen muß die Betriebssoftware zuerst von einem externen Speicher (Platte, Band, Diskette, Kassette) eingelesen werden. Dazu ist aber auch schon ein kleines Programm erforderlich. Dieses bezeichnet man als Urlader (IPL = Initial Program Loader), das irgendwo im System existiert oder von Hand einzugeben ist.

2. Allgemeines zum LASER 110-310 Betriebssystem

Das Betriebssystem der LASER-Computer 110-310 und des VZ200 ist ein in sich abgeschlossenes Programmpaket, das selbständig ablauffähig ist (stand alone system).

Es beinhaltet einen BASIC-Interpreter als Sprachumsetzer sowie die erforderlichen Unterstützungs- und Hilfsroutinen zur Ausführung von BASIC-Programmen. Darüberhinaus bietet es die Möglichkeit, Programme auf Kassette zu speichern und von Kassette wieder zu laden.

Das Disketten-Betriebssystem ist nur bei angeschlossener Diskettensteuerung verfügbar. Es erweitert den Sprachumfang des BASIC-Interpreters um Anweisungen zur Diskettenbearbeitung und bietet die Möglichkeit, Programme auf Diskette zu speichern und wieder von dort zu laden.

Ist ein Diskettenlaufwerk angeschlossen, so wird das Disketten-Betriebssystem beim Einschalten des Rechners mit dem internen Betriebssystem verknüpft.

Das Disketten-Betriebssystem hat einen eigenen Interpreter zur Erkennung und Bearbeitung der zusätzlichen Befehle, macht aber intensiven Gebrauch von Unterstützungs- und Hilfsroutinen des internen Systems und ist daher keine selbständig ablauffähige Programmeinheit.

EXTENDED BASIC ist eine Spracherweiterung des BASIC-Interpreters und bietet den Komfort zusätzlicher Befehle, vor allem im Grafik-Bereich und bei der Programm-Entwicklung. Es ist je nach Ausbaustufe Ihres Rechners auf Kassette oder Diskette verfügbar und muß zusätzlich zum internen Betriebssystem in den Rechner geladen werden.

3. Die Speicheraufteilung

```
   0H ----------------------------
      !                          !
      !        Internes          !
      !     Betriebssystem       !
      !                          !
4000H ----------------------------
      !                          !
      !                          !
      !      nicht belegt        !
      !                          !
      !                          !
6800H ----------------------------
      !   Ein-/Ausgabebereich    !
7000H ----------------------------
      !       Bildschirm-        !
      !        speicher          !
7800H ----------------------------
      !     Kommunikations-      !
      !        bereich           !
7AE9H ----------------------------
      !                          !
      !                          !
      !                          !
      !    freier Speicher       !
      !         (RAM)            !
      !                          !
      !                          !
 max. !                          !
FFFFH ----------------------------

            ohne DOS
```

```
   0H ----------------------------
      !                          !
      !        Internes          !
      !     Betriebssystem       !
      !                          !
4000H ----------------------------
      !                          !
      !       Disketten          !
      !     Betriebssystem       !
6000H ----------------------------
      !      nicht belegt        !
6800H ----------------------------
      !   Ein-/Ausgabebereich    !
7000H ----------------------------
      !       Bildschirm-        !
      !        speicher          !
7800H ----------------------------
      !     Kommunikations-      !
      !        bereich           !
7AE9H ----------------------------
      !                          !
      !                          !
      !                          !
      !    freier Speicher       !
      !         (RAM)            !
      !                          !
      ----------------------------
 max. !   DOS-Arbeitsbereich     !
FFFFH ----------------------------

             mit DOS
```

Die ersten 16K sind ausschließlich dem internen Betriebssystem zugeordnet.

Wenn Sie sich die 6 Grundbausteine eines Betriebssystems vor Augen halten, so ergibt sich in etwa die nachstehend aufgeführte Lage in diesem Bereich.
Es wurde allerdings keine klare Trennung durchgeführt, einzelne Codeelemente, die von ihrer Funktion einem bestimmten Grundbaustein zuzuordnen sind, wurden wild im Speicher verstreut. Vor allem am Ende des ROM-Bereichs finden Sie eine Reihe von "Rucksäcken", die bei späteren Versionsänderungen dort angelegt wurden.

```
0000H ----------------------
      !   Geräte-Treiber   !
0700H ----------------------
      !   Arithmetische    !
      !  u. mathematische  !
      !     Routinen       !
1600H ----------------------
      !  Unterstützungs-   !
      !     routinen       !
1A00H ----------------------
      !     Monitor        !
1C00H ----------------------
      !      BASIC         !
      !   Interpreter      !
2C00H ----------------------
      !   Hilfsroutinen    !
4000H ----------------------
```

Ist ein Diskettensystem angeschlossen, sind die folgenden 8K durch das Disketten-Betriebssystem belegt, ansonsten ist der Adreßbereich bis 6800H nicht mit Speicherbausteinen belegt.

Es folgt ein Bereich, der mit 'Ein-/Ausgabebereich' betitelt ist. Hinter diesem Adreßraum verbirgt sich kein Speicherbaustein. Bei entsprechender Adressierung werden vielmehr direkte Schnittstellen zu Ein-/Ausgabebausteinen angesprochen. So ist z.B. die Tastatur als Matrix in diesem Adreßraum verdrahtet, wobei jede einzelne Taste direkt abfragbar ist.

Es folgt der Bildschirmspeicher mit 2K. Dahinter verbirgt sich ein 2K RAM-Baustein, der vom Bildgenerator ständig ausgelesen und dargestellt wird.

Im darauffolgenden Kommunikationsbereich werden vom BASIC-Interpreter Arbeitsbereiche, Adreßzeiger und Verwaltungstabellen angelegt und benutzt.

Hinter dem Kommunikationsbereich liegt der Freispeicher. Das ist der Bereich, in dem vom Betriebssystem BASIC-Programme und zugehörige Variable gespeichert werden. Dort können Sie auch Ihre Assembler- oder Maschinenprogramme laden und zum Ablauf bringen.

Bei angeschlossenem Diskettenlaufwerk wird bei der Systeminitialisierung am Speicherende vom DOS ein 310 Byte langer Arbeitsbereich angelegt, der für das DOS ähnliche Funktionen wie der Kommunikationsbereich beim BASIC erfüllt. Ist kein Diskettenlaufwerk angeschlossen, reicht der Freispeicher bis zum RAM-Ende.



4. Der Ein-/Ausgabebereich (6800H - 6FFFH)

Dieser Speicherbereich dient der direkten Kontrolle von Ein-/Ausgabebausteinen wie Tastatur, Lautsprecher, Kassette und Bildgenerator.

Bei Eingabe- oder Ausgabeadressierung haben die jeweiligen Bits unterschiedliche Bedeutung.

Über diese Adressen eingelesen werden die Tastaturmatrix, die Kassetteneingabe und der vertikale SYNC-Puls des Bildgenerators.

Ausgabeseitig werden über diesen Adreßbereich der Bildgenerator, der Lautsprecher und die Kassette gesteuert und angesprochen. Da ein ausgegebener Wert nicht wieder zurückgelesen werden kann, hält das Betriebssystem stets eine aktuelle Kopie des Ausgabewertes in der Adresse 783BH des Kommunikationsbereichs und benutzt diese als Referenz, falls eine weitere Ausgabe erfolgen soll.

Außer bei der Tastatureingabe ist es belanglos, welche Adresse des Adreßraumes Sie ansprechen, es werden nur die oberen 5 Adreßbits ausgewertet.

Eingabe

```
       7   6   5   4   3   2   1   0        Bit
     !===!===!===!===!===!===!===!===!
       !   !  ---------------------
       !   !           !
 Vertikal  Kassette    Tastatur
   SYNC
   Puls
```

Ausgabe



Tastatur-Eingabe

Die Tastatur bildet in dem Adreßraum 6800H bis 68FFH eine Matrix von 8 Zeilen und 6 Spalten, die wie folgt organisiert ist:

A8 - A15 = 68

A7	A6	A5	A4	A3	A2	A1	A0	D0	D1	D2	D3	D4	D5	
0	1	1	1	1	1	1	1	H	L	:	K	;	J	= 687FH
1	0	1	1	1	1	1	1	Y	O	Ret	I	P	U	= 68BFH
1	1	0	1	1	1	1	1	6	9	-	8	0	7	= 68DFH
1	1	1	0	1	1	1	1	N	.		,	Spc	M	= 68EFH
1	1	1	1	0	1	1	1	5	2		3	1	4	= 68F7H
1	1	1	1	1	0	1	1	B	X	Shft	C	Z	V	= 68FBH
1	1	1	1	1	1	0	1	G	S	Ctrl	D	A	F	= 68FDH
1	1	1	1	1	1	1	0	T	W		E	Q	R	= 68FEH

Sie sehen, daß eine "0" in einem Adreßbit die auszulesende Zeile bestimmt. Ist eines der gelesenen Datenbits = 0, so ist die zugehörige Taste im Schnittpunkt der Matrix gedrückt.

Werden mehr als ein Adreßbit = 0 gesetzt, so werden auch gleichzeitig mehrere Zeilen ausgelesen. Mit der Adressierung von 6800H können Sie somit die Matrix auf einmal auslesen.

Kassetten-Eingabe

Über Bit 6 der Adresse 6800H (oder irgendeiner anderen des Bereichs 6800H-6FFFH) werden die Pulse vom Kassettenrekorder eingelesen.

Bildschirmsteuerung

Über den Adreßbereich 6800H-6FFFH können Steuerinformationen an den Bildgenerator übertragen werden.

Bit 3 = Bildschirmmodus
- 0 = Text
- 1 = Grafik

Bit 4 = Darstellungsfarbe
- 0 = grün
- 1 = rot

Lautsprecherausgabe

Mit den Bits 0 und 5 kontrollieren Sie den eingebauten kleinen Lautsprecher. Die beiden Bits müssen immer komplementär sein, sonst erzeugen Sie nie einen Ton (also Bit 0=1 und Bit 5=0 oder umgekehrt).
Durch das Umschalten dieser Bits in einer bestimmten Frequenz wird die Tonhöhe bestimmt.

Kassettenausgabe

über die Bitpositionen 1 und 2 des Adreßraumes 6800H-6FFFH geben Sie Datenbits auf die Kassette aus, wobei beide Bits stets gleich sein sollten.

Bei allen drei Ausgaben (Bildgenerator, Lautsprecher, Kassette), müssen Sie stets den vorherigen Status beachten und nur die Bits ändern, die Sie ändern wollen. Daher wird von den ROM-Routinen in Adresse 783BH des Kommunikationsbereichs stets eine aktuelle Kopie des Ausgabebytes gehalten. Sie sollten das auch beachten und nachvollziehen, falls Sie einmal direkt etwas an diese Bausteine ausgeben wollen.

Vertikal SYNC-Pulse

Der vom Bildgenerator u.a. erzeugte Vertikal SYNC-Puls (bei PAL alle 20 ms) wird normalerweise benutzt, um in der CPU einen Interrupt zu erzeugen. Wie später noch eingehend beschrieben, wird dieser Interrupt zur Synchronisation der übertragung in den Video-RAM benutzt, um ein flackerfreies Bild zu erhalten.

Dieser Puls ist auch über Bit 7 des Adreßbereichs 6800H-6FFFH tastbar, selbst wenn die Interrupts ausgeschaltet (disabled) sind. Sie können diesen dazu benutzen, innerhalb Ihrer Maschinenprogramme auch bei ausgeschalteten Interrupts eine Bildschirm-Synchronisation durchführen zu können, ohne eine eigene Interrupt-Behandlung programmieren zu müssen.



5. Der Bildschirmspeicher

Im Bildschirmspeicher sind die auf dem Bildschirm darzustellenden Zeichen einzutragen. Dieser Speicherbereich wird vom Bildgenerator ständig abgetastet und die Bildinformation 1:1 auf dem Bildschirm dargestellt. Der Bildschirmspeicher belegt den Adreßraum 7000H-77FFH (= 2 K).

Im Textmodus werden nur die ersten 512 Bytes benutzt, wobei jedes Byte ein darzustellendes Zeichen aufnehmen kann. Dies entspricht einer Ausgabekapazität von 16 Zeilen a' 32 Zeichen.

Beachten Sie, daß eigene Bildschirmcodes verwendet werden, die nicht immer dem ASCII-Code entsprechen.

Zwei Darstellungsfarben sind über die Bildschirmsteuerung wählbar, dies sind grün und rot. Dazu gibt es wiederum zwei Darstellungsarten, wobei die ausgewählte Farbe als Hintergrundfarbe oder als Schriftfarbe Verwendung findet. Die Steuerung der Darstellungart erfolgt über die Adresse 7818H des Kommunikationsbereichs. Zwischen den Darstellungsarten kann auch über die inverse Zeichendarstellung umgeschaltet werden.

Innerhalb der Textausgabe ist auch eine Blockgrafik in 8 verschiedenen Farben möglich.

Zeichentabelle:

00 = @	0B = K	16 = V	21 = !	2C = ,	37 = 7
01 = A	0C = L	17 = W	22 = "	2D = -	38 = 8
02 = B	0D = M	18 = X	23 = #	2E = .	39 = 9
03 = C	0E = N	19 = Y	24 = $	2F = /	3A = :
04 = D	0F = O	1A = Z	25 = %	30 = 0	3B = ;
05 = E	10 = P	1B = [	26 = &	31 = 1	3C = <
06 = F	11 = Q	1C = \	27 = '	32 = 2	3D = =
07 = G	12 = R	1D =]	28 = (	33 = 3	3E = >
08 = H	13 = S	1E = ^	29 =)	34 = 4	3F = ?
09 = I	14 = T	1F = _	2A = *	35 = 5	
0A = J	15 = U	20 =	2B = +	36 = 6	

Die Codes 40H bis 7FH stellen die gleichen Zeichen invers dar.

Ein Blockgrafik-Zeichen wird durch Bit7 = 1 gekennzeichnet.
Die Bitpositionen 4, 5 und 6 bestimmen eine von acht Farben

Bit 6	5	4		
0	0	0	=	grün
0	0	1	=	gelb
0	1	0	=	blau
0	1	1	=	rot
1	0	0	=	beige
1	0	1	=	hellgrün
1	1	0	=	rosa
1	1	1	=	orange

In den Bitpositionen 0 - 3 wird die Form der Blockgrafik bestimmt.

Jede der vier Bitpositionen ist einem Viertel einer Zeichenposition zugeordnet. Die nachstehende Abbildung stellt die Zuordnung der einzelnen Bits dar:

```
-----------
! 3 ! 2 !
-----------
! 1 ! 0 !
-----------
```

Im Grafik-Modus werden die gesamten 2K des Bildschirmspeichers benutzt, um eine Auflösung von 128 x 64 Punkten (Pixels) zu erreichen.

In jedem Byte wird die Information von vier Pixels gespeichert, wobei jedem einzelnen Pixel 2 Bits zugeordnet sind.

Es lassen sich mit diesen 2 Bits vier verschiedene Farben darstellen, wobei über die Bildschirmsteuerung (Darstellungsfarbe) zwischen zwei Farbsätzen gewählt werden kann.

6800H		Bit 4 = 0	Bit 4 = 1	
	00	grün	weiß	(Hintergrund)
	01	gelb	türkis	
	10	blau	orange	
	11	rot	violett	

6. Der Kommunikationsbereich

Der Kommunikationsbereich ist der Merkzettel des Betriebssystems und befindet sich im Adreßbereich 7800H bis 7AE9H.

In ihm befinden sich Tabellen, Zeiger und Adressen, die vom Betriebssystem dort angelegt und verwaltet werden.

Darüberhinaus sind dort Arbeitsbereiche und Zwischenspeicher-Bereiche vorhanden, die bei der Durchführung arithmetischer Operationen und Ein-/Ausgabe-Operationen benötigt werden.

Innerhalb des Betriebssystems befinden sich eine ganze Reihe von "RAM-Erweiterungsausgängen". Das sind CALL-Aufrufe auf einen 3-Byte Bereich im Kommunikationsbereich. Normalerweise sind die Ausgänge im Kommunikationsbereich mit einem RETurn belegt.

über die RAM-Erweiterungsausgänge lassen sich eigene Anschlußroutinen an die verschiedenen Betriebssystemfunktionen anschließen. EXTENDED BASIC nutzt dies sehr intensiv, auch das DOS schließt sich über einen solchen Erweiterungsausgang an den BASIC-Interpreter an.

Einige Hilfsroutinen, die jeweils vor dem Aufruf modifiziert werden müssen, wurden ebenfalls im Kommunikationsbereich untergebracht. Dorthin gebracht werden sie nach dem Einschalten des Rechners während der Systeminitialisierung aus dem ROM. Eine dieser Routinen ist ein Unterprogramm der Divisions-Funktion. Diese wird aus der ROM-Routine zur Division modifiziert und aufgerufen, um spezielle Subtraktionen und Vergleiche durchzuführen.

Eine Beschreibung jedes einzelnen Feldes des Kommunikationsbereichs finden Sie im Kapitel 9 "Adressen und Tabellen des BASIC-Interpreters".

7. Der Freispeicher

Der Freispeicher steht Ihnen zur Verfügung, eigene BASIC- und Maschinenprogramme zu laden und ablaufen zu lassen.

Nach dem Einschalten des Rechners erstreckt sich der Freispeicher vom Ende des Kommunikationsbereichs bis zum physikalischen Speicherende, bzw. bis zum Beginn des DOS-Arbeitsbereichs bei aktiviertem Diskettenlaufwerk.

Die Grenzen des Freispeichers werden in zwei Adreßzeigern des Kommunikationsbereichs angezeigt:

78A4H/78A5H enthält die Anfangsadresse
78B1H/78B2H enthält die Endadresse

Durch Manipulation dieser Zeiger können Sie den Freispeicher von oben oder von unten her verkleinern. Das bietet sich dann an, wenn man einen bestimmten Speicherbereich vor dem BASIC-Interpreter schützen will (siehe DOS-Arbeitsbereich).

Nutzen Sie den Freispeicher ausschließlich für Maschinenprogramme, so steht Ihnen der gesamte Raum uneingeschränkt zur Verfügung, das Maschinenprogramm ist für die Verwaltung dieses Raumes selbst verantwortlich.

Bei BASIC-Programmen übernimmt der BASIC-Interpreter des ROM das Speicher-Management für den Freispeicher-Bereich. Dieser wird in verschiedene Teilbereiche und Strukturen unterteilt und dynamisch verwaltet.

```
                !                        !
                !   Kommunikations-      !        PST = Program
                !      bereich           !              Statement
(78A4H) ==>     --------------------------              Table
                !        BASIC           !
                !   Programm-Tabelle     !        VLT = Variable
                !        (PST)           !              List Table
(78F9H) ==>     --------------------------
                !        BASIC           !        FSL = Free Space
                !  Variablen-Tabelle     !              List
                !        (VLT)           !
(78FDH) ==>     --------------------------
                !                        !
                !                        !
                !                        !
                !     Freispeicher       !
                !        (FSL)           !
                !                        !
                !                        !
(78E8H) ==>     --------------------------
                !        Stack           !
(78A0H) ==>     --------------------------
                !                        !
                !    String-Bereich      !
                !                        !
(78B1H) ==>     --------------------------
```

Beachten Sie, daß den oben gezeigten Bereichen keine festen Adressen zugeordnet sind. Diese Bereiche sind dynamisch, d.h. ihre Größen und Grenzen sind fließend und werden ständig dem aktuellen Bedarf angepaßt.

Betrachten Sie die Programm-Tabelle. Diese enthält das geladene BASIC-Programm. Wenn Sie dort eine Programmzeile einfügen, so dehnt sich die Programm-Tabelle aus, gleichzeitig verschiebt sich der Anfang der Variablen-Tabelle, die direkt an die Programm-Tabelle anschließt. Definieren Sie in Ihrem Programm eine neue Variable, so vergrößert sich die Variablen-Tabelle.

Da die Grenzen der einzelnen Bereiche fließend sind, werden ihre aktuellen Adressen in Adreßzeigern des Kommunikationsbereichs geführt. Dies erlaubt ein Verschieben der Tabellen entsprechend den Erfordernissen des Programms und bietet gleichzeitig dem Nutzer die Möglichkeit der Manipulation.

Die Programm-Tabelle (PST) enthält in komprimierter Form die einzelnen Zeilen des BASIC-Programms. Komprimiert bedeutet dabei, daß innerhalb der Zeilen die BASIC-Schlüsselworte durch 1-Byte Hexadezimal-Werte (TOKEN) ersetzt sind.

Die Startadresse der Programm-Tabelle ist im Adreßzeiger 78A4H des Kommunikationsbereichs enthalten. Nach dem Laden eines Programms bleibt diese konstant. Wird der Anfang des Freispeichers nicht manipuliert (EXTENDED BASIC tut so etwas z.B.), so werden Sie dort den Eintrag 7AE9H vorfinden.

Die Endadresse der Programm-Tabelle variiert mit der Größe des Programms. Sie ist identisch mit der Anfangsadresse der Variablen-Tabelle und ist im Adreßzeiger bei 78F9H zu finden.

Die Variablen-Tabelle (VLT) enthält Namen und Werte aller in einem BASIC-Programm definierten Variablen. Sie ist in zwei Abschnitte unterteilt, Abschnitt 1 enthält die einfachen Variablen und Abschnitt 2 dimensionierte Variable, d.h. Matrizen.

Variablen-Namen und -werte werden in der Reihenfolge in die Variablen-Tabelle eingetragen, wie sie beim Ablauf eines Programms auftreten. Jede neu auftretende Variable vergrößert die Tabelle. Einmal definierte Variable verbleiben bis zur Neuinitialisierung des Systems in der Tabelle, d.h. man kann sie nicht streichen, sondern höchstens ihren Wert ändern.

Vom Ende der Variablen-Tabelle bis zum Beginn des Stack-Bereichs befindet sich der verbliebene Rest des Freispeichers, der sich ja ursprünglich vom Ende des Kommunikationsbereichs bis zum Speicherende erstreckte. Sie sehen aus dem Bild, daß dieser Freispeicher von oben (unterer Adreßraum) durch die Programm-Tabelle und die Variablen-Tabelle und von unten (oberer Adreßraum) durch den String-Bereich und den Stack-Bereich eingeengt wird. Das geht solange gut, bis das Ende der Variablen-Tabelle mit dem Stack-Bereich kollidiert. In einem solchen Fall wird die Fehlermeldung "OUT OF MEMORY" ausgegeben.

Der Stack-Bereich ist sehr dynamisch. Er dient dem Betriebssystem ständig als Zwischenspeicher für Rücksprungadressen und Registerinhalte. Jeder CALL-, PUSH- oder RST-Befehl vergrößert den Stack-Bereich um 2 Byte, jeder RET- oder POP-Befehl verkleinert den Stack-Bereich entsprechend.

Vom BASIC-Interpreter werden darüberhinaus noch ganze Speicherblöcke auf den Stack geschrieben. Jede FOR-/NEXT-Schleife legt einen 18-Byte Block auf dem Stack an und jeder GOSUB-Befehl einen solchen von 7 Byte.

Der letzte dargestellte Abschnitt hinter dem Stack-Bereich ist der String-Bereich. Mit Ausnahme der im Programm fest definierten Textvariablen (in Anführungszeichen im Programmtext enthalten), werden alle Textvariablen (=Strings) in diesem Bereich abgelegt. Die Variablen-Tabelle enthält für Text-Variable als Wert die Verweisadresse auf den String-Bereich oder die Programm-Tabelle, wo der entsprechende Text zu finden ist.

Die Länge des String-Bereichs wird bei der Initialisierung mit 50 Byte definiert. Mit dem CLEAR-Kommando kann man diese jedoch variieren.

8. Funktionen des Betriebssystems

In den vorausgegangenen Abschnitten war schon mehrfach die Rede von den Grundbausteinen eines Betriebssystems. Nimmt man diese Aufgliederung als Anhalt, so läßt sich das Betriebssystem des LASER in die folgenden sieben Funktionsbereiche unterteilen:

1. System-Initialisierung
2. Eingabe-Routine
3. Interpretation und Ausführungssteuerung
4. Befehls- und Kommandoausführung
5. Arithmetische und mathematische Routinen
6. Ein-/Ausgabe - Treiber
7. System - Funktionen

Jeder dieser Funktionsbereiche soll jetzt etwas näher durchleuchtet werden.

System-Initialisierung

Im ersten Abschnitt dieses Buches wurde erwähnt, daß das Betriebssystem des LASER sich in ROM-Bausteinen befindet und sofort nach dem Einschalten verfügbar ist. Das ist nur die halbe Wahrheit. Zwar benötigt dieses System keine zusätzlichen Komponenten, die von irgendwoher nachzuladen sind, es sind jedoch einige Initialisierungsfunktionen durchzuführen, wie z.B. die Einrichtung des Kommunikationsbereichs, ehe mit dem Betriebssystem gearbeitet werden kann.

Was geschieht nun bei der System-Initialisierung ???

Nach dem Einschalten beginnt der Rechner seine Programmausführung immer bei der absoluten Adresse 0 des ROM-Bereichs. Dort wird der Bildgenerator in den Textmodus geschaltet und zur Adresse 674H verzweigt.

Bei 674H wird der ROM-Inhalt von Adresse 6D2H bis Adresse 707H in den Kommunikationsbereich von 7800H bis 7835H übertragen. Dies initialisiert die Adressen für die Restart-Vektoren 8, 10, 18 und 20, sodaß von dort ein Sprung in die aktuellen ROM-Routinen erfolgen kann. Die Restart-Vektoren 28, 30 und 38 werden mit RETurn-Befehlen belegt (keine Funktion).

Es werden ferner die "Geräte-Steuer-Blöcke" (Device Control Blocks) für Tastatur, Bildschirm und Drucker mit Ausgangswerten gefüllt (7815H - 782CH). Der Speicherbereich 7836H bis 785CH wird gelöscht (00H).

Zur weiteren Initialisierung wird zur Adresse 75H verzweigt. Dort werden folgende Funktionen ausgeführt:

Der ROM-Bereich von 18F7H bis 191DH wird in den Kommunikationsbereich von 7880H bis 78A6H übertragen. In diesem Bereich befindet sich u.a. die Hilfsroutine für die Division. Es werden ferner einige Schalter und Zeiger initialisiert, u.a. die Startadresse der Programm-Tabelle bei 78A4H.

Der Ein-/Ausgabepuffer ab Adresse 79E8H wird initialisiert, indem seine Anfangsadresse in den Pufferzeiger 78A7H eingetragen und vor den Puffer ein Vorspann von 3AH-00H-2CH geschrieben wird. Dieser Puffer dient u.a. zur Zwischenspeicherung jeder ein- und auszugebenden Programmzeile.

Adress-Vektoren für spezielle Diskettenbefehle von Adresse 7952H bis 79A5H werden mit einem Sprungbefehl zur Adresse 01D2H (JP 01D2H) vorbelegt. Dies verursacht bei Ansprung eines dieser Vektoren die Fehlermeldung "DISK COMMAND ERROR".
Diese Disk-Vektoren sind ein Relikt aus einem früheren Einsatzrechner dieses Betriebssystems und werden vom LASER-Computer nicht benutzt.

Die RAM-Erweiterungsausgänge 78A6H bis 79E2H werden mit RETurn (C9H) vorbesetzt.

Adresse 7AE8H (direkt vor dem Freispeicher) wird mit 00H geladen und der Stack-Zeiger wird zunächst mit 79F8H (im Ein-/Ausgabepuffer) initialisiert. Dies geschieht, da nun die Initialisierung mit CALL-Aufrufen fortgesetzt wird und dabei ein Stack zum Abspeichern der Rücksprungadresse benötigt wird.

Eine Unterroutine bei 1B8FH (innerhalb der NEW-Kommandoroutine) wird aufgerufen. Dort wird der Stack-Zeiger auf 7B45H umgesetzt, eine Adresse im Freispeicher-Bereich.

Der String-Zwischenspeicher (ab Adresse 78B5H) wird als leer gekennzeichnet, indem der Adreßzeiger bei 78B3H auf den ersten Eintrag gesetzt wird.

Als Ausgabegerät wird der Bildschirm gesetzt und, wenn erforderlich, ein Wagenrücklauf auf einen angeschlossenen Drucker ausgegeben.

Die Indizierungssperre wird zurückgesetzt und das Ende des Stack-Bereichs mit 0 gekennzeichnet.

Der Bildspeicher wird gelöscht (CLS).

Die Speicherendadresse wird ermittelt und in 78B1H gespeichert. Für den String-Bereich werden 50 Byte am Ende des Freispeichers reserviert und die Anfangsadresse des String-Bereichs in 78A0H eingetragen.

Die Unterroutine 1B4DH (NEW) wird aufgerufen.
Dort wird die TRACE-Funktion und der AUTO-Modus ausgeschaltet.
Der Anfang der Programm-Tabelle wird mit 00H-00H initialisiert, um diese als leer zu kennzeichnen.
Der Adreßzeiger auf die Variablen-Tabelle (=Endadresse der Programmtabelle) bei 78F9H wird auf die Anfangsadresse der Programmtabelle + 2 gesetzt. Da die Variablen-Tabelle auch noch leer ist, werden die Endadressen beider Teile bei 78FBH und 78FDH auf den gleichen Wert gesetzt.
Die TypCode-Tabelle ab 7901H wird für alle Variablen auf "einfache Genauigkeit" gesetzt.
Der Stack-Bereich wird vor dem String-Bereich eingerichtet.

Im Anschluß daran wird eine Unterroutine bei 3484H aufgerufen.
Dort wird geprüft (CALL 3AF0H), ob gleichzeitig mit dem Einschalten die CTRL-Taste gedrückt war (grüner Hintergrund) und entsprechend die Hintergrund-Flags 7818H und 7819H gesetzt.
Zur Erzeugung des richtigen Hintergrunds wird der Bildschirm nochmals gelöscht.

Über die Ein-/Ausgabeadressen (6800H) wird die Grundeinstellung 20H gesetzt und auch im Speicher bei 783BH vermerkt. Dies bedeutet "Darstellungsfarbe = grün" und "Textmodus".

Die Grundwerte für den Verzögerungszähler (783AH) und den Blinkzähler (7841H) werden gesetzt.

Als Farbwert wird "gelb" gesetzt (7846H) und der Interrupt-Vektor in 787DH auf RETurn (C9H).

Jetzt erfolgt endlich die Ausgabe des Vorstelltextes "VIDEO TECHNOLOGIE ...." und es wird der Z80 auf Interrupt-Modus 1 geschaltet. Das bedeutet, daß bei maskierbaren Interrupts die Programmadresse 38 angesprungen wird. Ein solcher Interrupt erfolgt, wenn Interrupts zugelassen sind (EI), alle 20 Millisekunden.

Ab Adresse 068EH findet die letzte Phase der Initialisierung statt. Dort wird nacheinander geprüft, ob bei den Adreßbereichen 4000H, 6000H oder 8000H ein ROM-Kassetteneinschub vorhanden ist (z.B. das DOS ab Adresse 4000H). Ein solcher Kassetteneinschub muß bei den o.a. Adressen mit der Bytefolge AAH-55H-E7H-18H beginnen.
Wird diese Bytefolge bei einer der Adressen ermittelt, so wird auf die darauf folgende Adresse verzweigt, ansonsten wird das Programm mit der BASIC-Eingaberoutine fortgesetzt.

Auf diese Art wird u.a. festgestellt, ob ein Diskettensystem angeschlossen ist. Die Disketten-Zusatzroutinen sind in ROMs in der Diskettensteuerung untergebracht. Die Adressierung beginnt bei Adresse 4000H, wobei die ersten vier Speicherstellen mit AAH-55H-E7H-18H belegt sind. Ist das Diskettensystem angeschlossen, so wird diese Bytefolge bei 4000H gefunden und das Programm zur Initialisierung des DOS ab Adresse 4004H fortgesetzt.

Die Eingabe-Routine

Die Eingabe-Routine ist in allen Betriebssystemen ähnlich. Ihre Funktion ist die Übernahme von Tastatureingaben und das Reagieren auf empfangene Kommandos.

Im Betriebssystem der LASER-Rechner 110, 210, 310 und des VZ200 werden sowohl System-Kommandos als auch BASIC Programmzeilen von der Eingabe-Routine verarbeitet.

Der Einsprung in die Eingabe-Routine erfolgt bei Adresse 1A19H, auch als Beginn der BASIC-Hauptschleife oder BASIC-Warmstart-Adresse bekannt.

Es wird dort zunächst die Meldung "READY" ausgegeben und über den RAM-Erweiterungsausgang bei 78ACH gesprungen (initialisiert mit RETurn).

Die Funktionen der Eingabe-Routine können wie folgt unterteilt werden:

1. Zeile von der Tastatur einlesen
2. BASIC-Schlüsselworte in der Zeile durch TOKEN ersetzen
3. Prüfen, ob ein Direktkommando eingegeben wurde
   (BASIC-Zeile ohne Zeilennummer).
   Wenn ja, weiter bei 6.
4. Zeile in Programmtabelle übertragen.
5. zurück zu 1.
6. Interpretieren und ausführen

Die Eingabe-Schleife beginnt bei der Adresse 1A33H. Ist das System im AUTO-Modus, wird zunächst die Zeilennummer und, wenn diese bereits im Programm vorhanden ist, auch der Zeileninhalt ausgegeben.

Mit einem CALL zur Routine bei 03E3H wird eine Zeile von der Tastatur übernommen und in den Ein-/Ausgabepuffer ab Adresse 79E8H übertragen. Wurde die Zeileneingabe mit der BREAK-Taste abgeschlossen, so erfolgt sofort Rücksprung zum Schleifenanfang (1A33H), d.h. die Zeileneingabe wird ignoriert.

Eine eingegebene Zeilennummer wird vom ASCII- in das Binärformat umgewandelt (CALL 1E5AH).

In der Unterroutine ab 1BC0H wird die Eingabezeile untersucht und alle darin enthaltenen BASIC-Schlüsselworte durch 1-Byte hexadezimale Kenner, die "TOKENs", ersetzt.

Danach erfolgt ein Sprung zum RAM-Erweiterungsausgang bei 79B2H, eine gute Gelegenheit, wenn man selbst noch die Eingabezeile behandeln möchte (Anschluß einer Maschinenroutine). Hier wurde u.a. EXTENDED BASIC zur Erkennung und Umsetzung der zusätzlichen Befehle eingehängt.

Bei 1AA4H wird geprüft, ob eine Zeilennummer eingegeben wurde. Wenn nicht, handelt es sich um ein Direktkommando. In diesem Fall wird sofort zur Interpretation und Ausführungssteuerung (1D5AH) verzweigt.

Ist eine Zeilennummer vorhanden, so wird die eingegebene Zeile an der richtigen Stelle der Programmtabelle hinzu- oder eingefügt.

Existiert bereits eine Zeile gleicher Nummer, so wird diese zuvor mit der Routine bei 2BE4H gelöscht.

Die Endadresse der Programmtabelle (78F9H) wird um die Länge der eingegebenen Zeile erhöht (1AC2H....) und mit der Routine bei 1955H Platz für die neue Zeile geschaffen, indem die dahinter liegenden Zeilen mit höherer Zeilennummer nach oben im Speicher verschoben werden.

Ab 1AD0H beginnt die Vorbereitung und Übertragung der Zeile aus dem Ein-/Ausgabepuffer in die Programmtabelle.

Ist die eingegebene Zeile leer, so wird dies bei 1ABFH bemerkt und die Übertragungsroutine übersprungen.

Die Verweisadressen zur Zeilenverkettung werden im gesamten Programm von der Routine bei 1AFCH erneuert. 1B5DH in der NEW-Routine wird zum Abschluß noch aufgerufen, um die Variablentabelle zu löschen und einige Flags und Kenner zurückzusetzen, sodaß nach Einfügen oder Ändern einer Zeile keine Fortführung des Programms mit CONT mehr möglich ist.

Danach erfolgt ein Rücksprung zum Schleifenanfang, um die nächste Zeile einzulesen.

Der AUTO-Befehl ist nur im EXTENDED BASIC verfügbar, kann aber mit Hilfe eines POKE 30945,1 auch im normalen BASIC eingeschaltet werden, wenn man mit den Standardwerten (Anfangswert = 10, Schrittweite = 10) zufrieden ist.

Im AUTO-Modus wird die auszugebende Zeilennummer in Adresse 78E2H/78E3H bereitgestellt und der Erhöhungswert bei 78E4H.
Das Einlesen der Zeilen im AUTO-Modus erfolgt im Abschnitt 1A3FH bis 1A76H. Dabei wird zunächst die Zeilennummer aus 78E2H/78E3H ausgegeben und, wenn diese Zeile bereits vorhanden ist (Suche mit 1B2CH), mit 2E53H auch der Zeileninhalt.

Das Einlesen erfolgt auch über 03E3H. Wurde die BREAK-Taste betätigt, so wird neben dem Ignorieren einer evtl. Eingabe auch der AUTO-Modus ausgeschaltet. Nach dem Einlesen der Zeile (ab 1A60H) wird die bei 78E2H/78E3H gespeicherte Zeilennummer um den Wert aus 78E4H erhöht.

Die weitere Behandlung der Zeile erfolgt, wie bei der Eingabe ohne AUTO, ab 1A81H.

Anmerkung:
Beim LASER 110, 210, 310 und VZ200 sind eine Reihe von BASIC-Schlüsselworten nicht kodiert, obwohl die erforderlichen Ausführungsroutinen und TOKEN vorhanden sind (siehe Befehlstabelle ab 1650H). Wahrscheinlich ist dies aus urheberrechtlichen Gründen geschehen.
Mit EXTENDED BASIC werden einige dieser Befehle und Funktionen, neben vielen zusätzlichen, geöffnet und verfügbar gemacht.

Interpretation und Ausführungssteuerung

Die Ausführung von Systemkommandos und BASIC-Befehlen erfolgt durch Interpretation.

Interpretation bedeutet, daß alle Kommandos und alle BASIC-Zeilen erst zur Ausführungszeit vom Betriebssystem analysiert und die erforderlichen Operationen angestoßen werden.

So ein Verfahren ist auf der Systemkommando-Ebene allgemein üblich. Ein Kommando wird eingelesen, interpretiert und die entsprechende Ausführungsroutine angestoßen.

Ein Programm mit Hilfe der Interpretation auszuführen ist im allgemeinen auf Home- und Personal-Computer beschränkt und dort auch nur für wenige Sprachen, wie z.B. BASIC oder LOGO.

Die Alternative zur Interpretation ist das vorherige Kompilieren mittels eines Compilers.

Compiler übersetzen den Quell-Code, das sind die Eingabezeilen in der entsprechenden Sprache (FORTRAN, COBOL, PASCAL, PL1 usw.), in direkt ausführbaren Maschinencode, Objekt-Code genannt.

Ein solcher Objekt-Code wird zur Ausführungszeit mit Hilfe eines Laders (Teil des Betriebssystems) in den Speicher geladen und gestartet. Nach dem Start läuft ein solches Programm fast völlig unabhängig vom Betriebssystem.

Die LASER-Rechner 110, 210, 310 und der VZ200 arbeiten im BASIC ausschließlich mit der Methode der Interpretation. Wollen Sie direkt ausführbaren Maschinencode verwenden, so müssen Sie sich diesen mit einem Assembler erzeugen lassen oder direkt eingeben.

Das Hauptwerkzeug eines Interpreters ist zunächst der Vergleich. Eine eingegebene Programmzeile eines BASIC-Programms wird Zeichen für Zeichen überprüft und nach BASIC-Schlüsselworten, wie IF, THEN, FOR, NEXT, GOTO usw. durchsucht. Jedes gefundene Schlüsselwort wird durch eine eindeutige hexadezimale Ziffer ersetzt, die als TOKEN bezeichnet wird (z.B. CLS = 84H, IF = 8FH, GOSUB = 91H, CLOAD = B9H). Anschließend wird die so behandelte Programmzeile in die Programm-Tabelle übertragen.

Diese Funktion findet bereits bei der Programmeingabe statt und entlastet die Ausführungssteuerung von dieser mühevollen und zeitraubenden Arbeit.

Zum Ausführungszeitpunkt wird wieder Zeile für Zeile von der Ausführungssteuerung adressiert und auf das Vorhandensein dieser TOKEN untersucht. Für fast jeden TOKEN besteht im BASIC-Interpreter eine eigene Ausführungsroutine, die beim Auffinden des entsprechenden TOKEN zur Ausführung der dahinter verborgenen Funktion aufgerufen wird.

Diese Routinen zur Befehls- und Kommandoausführung führen dann zunächst eine weitere formale (syntaktische) Prüfung durch:

- stimmt die Anzahl der Parameter?
- wurden die richtigen Datentypen verwendet?
- stehen die Kommas an der richtigen Stelle?
- sind Parameter erforderlichenfalls in Klammern eingeschlossen?

usw.

Bei einem Compiler würden diese Funktionen zur Ausführungszeit alle entfallen, da sie bereits während der Kompilierung stattfinden.

Beim LASER wird die Ausführungssteuerung aufgerufen, wenn eine Kommando- oder Programmzeile ohne Zeilennummer eingegeben wurde oder nachdem ein RUN-Kommando erkannt wurde.

Bei einem RUN-Kommando wird ein vollständiges, in der Programm-Tabelle gespeichertes BASIC-Programm ausgeführt.

Die Ausführungssteuerung beginnt bei Adresse 1D1EH und endet bei Adresse 1D77H. Der Einsprung erfolgt bei Adresse 1D5AH.

Folgende Schritte werden bei Ausführung einer Programmzeile durchgeführt:

1. Erstes Zeichen der aktuellen Zeile aus der Programm-Tabelle laden.
   Ist das Ende der Programm-Tabelle erreicht, erfolgt Rücksprung zur Eingabe-Routine.

2. Ist der Ablaufverfolger eingeschaltet (TRON aktiv), so wird zunächst die Zeilennummer auf dem Bildschirm ausgegeben (<nn>).

3. Ist das Zeichen kein TOKEN, weiter bei 7.

4. Ist das Zeichen > 'BBH', so muß es genau 'FAH' (MID$) sein, sonst ist es am Zeilenanfang nicht erlaubt und es wird SYNTAX ERROR erzeugt.

5. Ist das Zeichen kleiner 'BCH', so wird es als Index für eine Sprungtabelle der Ausführungsroutinen genutzt.

6. Die entsprechende Ausführungsroutine wird aufgerufen und nach Ausführung derselben zu 1. zurückgesprungen.

7. Ist das Zeichen kein TOKEN, muß es sich um eine Wertzuweisung handeln.
   Die angegebene Variable wird ermittelt, wenn nicht vorhanden, wird sie neu in die Variablen-Tabelle aufgenommen.

8. Der Wertausdruck wird analysiert und der Wert der Variablen zugeordnet.

9. Zurück zu 1.

Anmerkung:

Beim LASER sind die Befehle TRON und TROFF nur im EXTENDED BASIC verfügbar. Sie können diese jedoch im normalen BASIC durch POKE-Befehle ersetzen.

```
POKE 31003,1   =   TRON
POKE 31003,0   =   TROFF
```

Die Ausführungsroutine beginnt also mit dem Laden des ersten Zeichens der zu bearbeitenden Zeile.

Es wird anschließend die Adresse 1D1EH auf den Stack gepackt. Das ist die Adresse, an die alle Ausführungsroutinen nach erfolgreichem Abschluß ihrer Operationen zurückkehren sollen.

Ist das gelesene Zeichen kein TOKEN (< 80H), so sollte es eine Wertzuweisung sein, z.B. B=5.

Die Routine zur Durchführung von Wertzuweisungen beginnt bei 1F21H. Das ist die gleiche Adresse, bei der man landet, wenn vor der Zuweisung das TOKEN 8CH für LET gestanden hätte. Aus diesem Grunde kann man auch das LET weglassen.

Die Zuweisungsroutine erwartet, daß der Adreßzeiger auf die zu bearbeitende Zeile direkt vor dem Variablen-Namen steht. Es wird die Variablen-Tabelle auf einen Eintrag gleichen Namens durchsucht, wenn nicht vorhanden, wird der Name neu in die Tabelle aufgenommen. Hinter dem Variablen-Namen muß sich ein Gleichheitszeichen befinden und danach der Wertausdruck. Der Wertausdruck wird in der Routine ab 2337H analysiert. Der ermittelte Wert wird anschliessend in den richtigen Typ der angegebenen Variablen umgewandelt und an der Variablenadresse gespeichert.

Wird am Anfang der Zeile ein TOKEN festgestellt, so wird geprüft, ob es ein gültiges TOKEN ist. Gültig sind am Anfang einer Anweisung nur die TOKEN 80H bis BBH. Die TOKEN BCH bis F9H können nur als Teil einer Wertzuweisung oder einer Befehlsfolge benutzt werden.

Beispiel: 8FH (IF) 'ausdruck' CAH (THEN) xxxx

Das THEN-TOKEN darf nicht am Anfang einer Zeile stehen, sondern nur nach einer IF-Anweisung.

Einzige Ausnahme dazu ist das MID$-TOKEN 'FAH', das aber beim LASER auch nicht so ohne weiteres verwendet werden kann, da es einen der unbenutzten RAM-Erweiterungsausgänge im Kommunikationsbereich benutzt. Hier lassen sich jedoch geschickt neue, selbst gemachte BASIC-Befehle an den Interpreter anschließen, z.B. ein SORT o.ä..

Ein TOKEN zwischen 80H und BBH wird als Index in die Sprungtabelle ab Adresse 1822H benutzt, in der für jede gültige Anweisung die Anfangsadresse der Ausführungsroutine gespeichert ist. Das Programm wird dann an der dort entnommenen Adresse fortgesetzt.

Die hinter dem TOKEN befindlichen Werte bis zum Anweisungsende sind die für die jeweilige Ausführungsroutine erforderlichen Parameter. Jede Ausführungsroutine kennt die zu erwartenden Parameter und prüft diese auf Vollständigkeit und korrektes Format. Das Ende der Parameter eines Befehls muß auch mit dem Ende der Anweisung übereinstimmen.

Nach Abschluß einer Ausführungsroutine wird die Kontrolle wieder der Ausführungssteuerung übergeben (1D1EH). Dort wird zunächst geprüft, ob das Ende der Anweisung erreicht ist. Das Ende einer Anweisung ist entweder eine Zeilenendekennung 00H oder ein Anweisungstrenner ':'. Bei Erreichen eines Anweisungstrenners wird die darauf folgende Anweisung in derselben Zeile auf die gleiche Art interpretiert und ausgeführt.

Bei Erreichen des Zeilenendes wird die nächste Zeile in der Programm-Tabelle adressiert und der Ausführungssteuerung übergeben.

Bei einem System-Kommando oder einem Direktbefehl gibt es jedoch keine "nächste" Zeile, diese Anweisungen werden auch nicht aus der Programm-Tabelle heraus, sondern direkt aus dem Ein-/Ausgabepuffer ausgeführt. Dort ist am Ende des Puffers ein Programmende 00H-00H simuliert. Zusätzliche wird noch zur Kennung einer solchen Anweisung eine Zeilennummer von FFH-FFH oder 65535 eingespielt.

Mit dem RUN-Befehl wird der Ausführungssteuerung mitgeteilt, daß sie ihre Anweisungen aus der Programm-Tabelle zu entnehmen hat.

Bei Erreichen des Endes eines BASIC-Programms, sei es mit oder ohne END-Anweisung, wird über die END-Ausführungsroutine zur Eingaberoutine zurückgesprungen.

Ein Fehler, der während der Ausführung festgestellt wird, verursacht die Ausgabe einer entsprechenden Fehlermeldung und ebenfalls die Rückkehr zur Eingabe-Routine.

Die Ausführungsroutinen

In den Ausführungsroutinen werden die eigentlichen Funktionen der einzelnen Befehle ausgeführt. Für jedes System-Kommando (CLOAD, CSAVE, CLEAR, RUN usw.) und für jeden BASIC-Befehl (FOR, IF, GOSUB, GOTO usw.) existiert eine gesonderte Ausführungsroutine. Zusätzlich sind Ausführungsroutinen für alle mathematischen Funktionen, wie SIN, COS, ATN, LOG usw. vorhanden.

Diese Ausführungsroutinen analysieren die Anweisung von der Stelle aus weiter, an der die Ausführungsteuerung ein TOKEN entdeckt hat. Die Anweisung wird von links nach rechts nach speziellen Zeichen, wie Kommas oder Klammern oder TOKEN untersucht. Jede Anweisung hat ihren speziellen Parameteraufbau, sodaß hier zunächst eine weitere formelle Prüfung stattfindet.

Die Ausführungsroutinen erfüllen in vielen Fällen auch wieder Steuerfunktionen, indem sie zur Erfüllung ihrer Funktion eine ganze Reihe interner Unterroutinen aufrufen, die sie sich mit anderen Ausführungsroutinen teilen.

Ein gutes Beispiel für eine solche interne Unterroutine ist die Ausdrucksanalyse bei 2337H. Diese Routine wird von allen Ausführungsroutinen aufgerufen, die Ausdrücke in ihren Parametern zulassen. Beispiele solcher Ausführungsroutinen sind die zur Bearbeitung von IF, FOR und PRINT.

Die Ausdrucksanalyse ruft weitere interne Unterroutinen auf, so z.B. 260DH um Variable innerhalb eines Ausdrucks zu bearbeiten. Da es auch indizierte Variable gibt, die als Indizierung wiederum einen Ausdruck zulassen, muß diese Routine ggf. wieder die Unterroutine zur Ausdrucksanalyse aufrufen, von der sie selber gerade aufgerufen wurde. So etwas bezeichnet man als Rekursion.

Ein Beispiel für eine Wertzuweisung, die eine solche Rekursion hervorruft:

ST = E(BC/CD(3,F))/D(DE-1)

Andere interne Unterroutinen sind :

- Vorsetzen ans Anweisungsende (1F05H)
- einen FOR- oder GOSUB-Datenblock auf dem Stack suchen (1936H)
- einen Eintrag in den String-Zwischenspeicher schreiben (2865H)

und viele andere mehr.

Zwischenresultate werden in der Regel in einem Arbeitsbereich des Kommunikationsbereichs (X-Register = 791DH) abgelegt.

Alle Ausführungsroutinen (außer MID$) werden mit folgenden Registerinhalten angesprungen:

A	-	das auf das TOKEN folgende Zeichen
Flags	-	CARRY = numerisch
		ZERO = Anweisungsende ':' oder Zeilenende X'00'
BC	-	Anfangsadresse der Ausführungsroutine
DE	-	Adresse des Sprungstabelleneintrags + 1 für das TOKEN
HL	-	Adresse des in A stehenden Zeichens in der Anweisung

Eine Tabelle aller Systemkommandos und BASIC-Befehle mit der Adresse ihrer Ausführungsroutinen ist im Kapitel 9 enthalten.

Arithmetische und mathematische Funktionen

Zunächst einige Betrachtungen der rechnerischen Fähigkeiten des Z80 und des BASIC-Interpreters.

Der Z80 unterstützt intern nur 8 Bit- und 16 Bit- Additionen und Subtraktionen. Er führt keine Multiplikationen oder Divisionen durch und erst recht keine Fließkomma-Arithmetik.

Der Registersatz des Z80 zur Durchführung der Arithmetik besteht aus sieben 16-Bit Registerpaaren (AF, BC, DE, HL, IX, IY, SP), hinzu kommen vier Schattenregisterpaare (AF', BC', DE', HL'), die allerdings nur zur Zwischenspeicherung verwendet werden können.

Alle arithmetischen Operationen müssen grundsätzlich zwischen diesen Registern stattfinden. Register-Speicher Operationen sind nur in wenigen Ausnahmefällen über indirekte Adressierung möglich.

Die Operationen der Register untereinander sind auch beschränkt. Vor allem in der 16-Bit Arithmetik sind nur ganz bestimmte Registerkonstellationen zulässig.

Der BASIC - Interpreter unterstützt alle arithmetischen Operationen, sei es Addition, Subtraktion, Multiplikation oder Division und dies für drei verschiedene Typen von Variablen:

- Ganzzahlige Variable (Integer)
- Variable einfacher Genauigkeit (single precision)
- Variable doppelter Genauigkeit (double precision)

Dies wird erreicht durch interne Unterroutinen, die die fehlenden Hardwarefähigkeiten des Z80 per Software ersetzen.

Durch wie Komplexität der Software bedingt werden allerdings keine Mischoperationen unterstützt, d.h. es können immer nur Variable gleichen Typs miteinander verknüpft werden. Die Verwendung ungleicher Variablentypen in einer Operation würde zu unvorhersehbaren Ergebnissen führen, sie hätten damit eine neue Art von Zufallsgenerator geschaffen.

Die drei Variablentypen, die der BASIC-Interpreter unterstützt, haben folgendes Format:

Ganzzahlige Variable:	16 Bits	(1 Bit Vorzeichen, 15 Bits Daten)
Variable einfacher Genauigkeit:	32 Bits	(8 Bits Exponent, 24 Bits Mantisse mit Vorzeichen)
Variable doppelter Genauigkeit:	56 Bits	(8 Bits Exponent, 48 Bits Mantisse mit Vorzeichen)

Hieraus wird ersichtlich, daß die Hardware-Register nicht ausreichen, um zwei Variable einfacher oder doppelter Genauigkeit aufzunehmen, geschweige denn zu verarbeiten.

Aus diesem Grunde wurden im Kommunikationsbereich zwei Arbeitsbereiche eingerichtet, die als Zwischen- und Arbeitsspeicher (Registerersatz) benutzt werden.

Dies sind der Arbeitsbereich 1 (als X-Register bezeichnet), der den Bereich 791DH bis 7924H belegt und der Arbeitsbereich 2 (Y-Register), der sich von 7927H bis 792EH erstreckt.

Diese beiden Arbeitsbereiche haben folgendes Format:

Adresse	Ganzzahlig	Einfache Genauigkeit	Doppelte Genauigkeit
791D	---	---	LSB
791E	---	---	NSB
791F	---	---	NSB
7920	---	---	NSB
7921	LSB	LSB	NSB
7922	MSB	NSB	NSB
7923	---	MSB	MSB
7924	---	EXP	EXP

(Die Adressen beziehen sich auf Arbeitsbereich 1, Arbeitsbereich 2 ist jedoch gleich strukturiert)

LSB = niederwertigstes Byte (least significant byte)
NSB = nächst höherwertiges Byte (next significant byte)
MSB = höchstwertigstes Byte (most significant byte)
EXP = Exponent

Die verschiedenen arithmetischen Operationen für die drei Variablentypen haben folgende Register-/Arbeitsbereichszuordnung:

Ganzzahlige Variable

1.Operand		2.Operand	Operation	Ergebnis	Ausf.routine
HL	+	DE	Addition	HL	0BD2H
HL	-	DE	Subtraktion	HL	0BC7H
HL	*	DE	Multiplikation	HL	0BF2H
DE	/	HL	Division	ARB1	2490H

Variable einfacher Genauigkeit

1.Operand		2.Operand	Operation	Ergebnis	Ausf.routine
ARB1	+	BCDE	Addition	ARB1	0716H
ARB1	-	BCDE	Subtraktion	ARB1	0713H
ARB1	*	BCDE	Multiplikation	ARB1	0847H
ARB1	/	BCDE	Division	ARB1	08A2H

Die beiden Registerpaare BC und DE werden bei Operationen einfacher Genauigkeit zur Aufnahme des 2. Operanden benutzt.

Variable doppelter Genauigkeit

1.Operand		2.Operand	Operation	Ergebnis	Ausf.routine
ARB1	+	ARB2	Addition	ARB1	0C77H
ARB1	-	ARB2	Subtraktion	ARB1	0C70H
ARB1	*	ARB2	Multiplikation	ARB1	0DA1H
ARB1	/	ARB2	Division	ARB1	0DE5H

Da gemischte Operationen nicht zugelassen sind, können ganzzahlige Werte nur mit anderen ganzzahligen Werten verarbeitet werden. Dasselbe gilt natürlich für Werte einfacher oder doppelter Genauigkeit.

Da für jeden Typ vier verschiedene arithmetische Operationen möglich sind (+ - * /) und es drei verschiedene Datentypen gibt, existieren somit auch zwölf verschiedene arithmetische Routinen. Hinzu kommen 3 Routinen für typabhängige arithmetische Vergleichsoperationen.

Eine Adreßtabelle der 15 Routinen ist im ROM ab Adresse 18ABH enthalten.

Jede dieser Routinen kennt den Typ der zu verarbeitenden Werte und erwartet, daß diese in den richtigen Registern bzw. Bereichen bereitstehen.

Das trifft aber nicht für die mathematischen Routinen zu, da diese nur mit einem Eingangswert arbeiten, der immer im Arbeitsbereich 1 (X-Register) bereitgestellt werden muß.

Für die mathematischen Routinen ergibt sich jetzt ein Problem. Sie müssen intern arithmetische Operationen aufrufen, können im Arbeitsbereich 1 aber nicht den Typ des dort bereitgestellten Arguments erkennen. Aus diesem Grunde wurde im Kommunikationsbereich ein weiteres Byte belegt (78AFH), das Auskunft über den Typ des im Arbeitsbereich 1 gespeicherten Wertes gibt. Diese Byte wird auch als Typ-Flag bezeichnet.

Es wird dort einer der vier folgenden Codes eingetragen:

Code	Daten-Typ
02	Ganzzahlig
03	Textvariable
04	einfache Genauigkeit
08	doppelte Genauigkeit

Der Typ-Code entspricht genau der Länge des im Arbeitsbereich 1 für den spezifischen Datentyp gespeicherten Wertes.

Die mathematischen Routinen lassen, bis auf wenige Ausnahmen, alle verschiedenen Datentypen zu (siehe detaillierte Beschreibung der Routinen).

Interne Darstellung der Daten

Zum besseren Verständnis des zuvor gesagten soll hier erläutert werden, wie die Daten im LASER intern eigentlich dargestellt werden.

Ganzzahlige Variable werden in 16 Bits dargestellt, wobei Bit 15 das Vorzeichen und die Bits 0 - 14 den Wert enthalten. Der dadurch größte darstellbare positive Wert ist 32767 (dezimal) oder 7FFF (hexadezimal). Der kleinste darstellbare negative Wert ist -32768 (dezimal) oder 8000 (hexadezimal).

```
Bit ==>  15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
        !--!--!--!--!--!--!--!--!--!--!--!--!--!--!--!--!
        !  -----------------------------------------------
        !
        !                       !

     Vorzeichen               Wert
    0 = positiv
    1 = negativ
```

Positive Werte: 0000 - 7FFF (hex) = 0 - 32767 (dez)
Negative Werte: FFFF - 8000 (hex) = -1 - -32768 (dez)

Beachten Sie, daß negative Werte in ihrem 1er Komplement dargestellt werden.

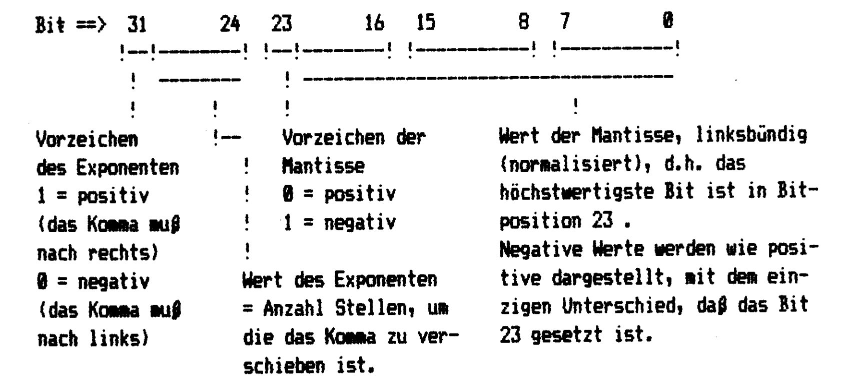
<u>Fließkomma-Variable</u> werden vom BASIC in zwei verschiedenen Typen verarbeitet:

- Variable einfacher Genauigkeit
- Variable doppelter Genauigkeit

Bei Typen haben einen acht Byte langen Exponenten mit Vorzeichen.

Bei Variablen einfacher Genauigkeit hat die Mantisse eine Länge von 24 Byte mit Vorzeichen und bei Variablen doppelter Genauigkeit eine Länge von 56 Byte mit Vorzeichen.

Beide Typen haben das gleiche Format, sie unterscheiden sich nur in der Mantissenlänge.



<u>Typ = einfache Genauigkeit</u>

Die Zahlendarstellung erfolgt in der Form

Zahl = Mantisse * 2^{EXP} mit 0.5 <= Mantisse < 1

Die Mantisse ist 24 oder 56 Bit lang, wobei das erste Bit nicht mit abgespeichert wird, da es immer 1 ist. An seine Stelle wird das Vorzeichen der Mantisse gesetzt.

Der Exponent wird immer mit einem Offset von 128 (= 80H) abgespeichert, daraus ergibt sich das Vorzeichen des Exponenten in der höchstwertigen Bitposition.

Beispiele: 0.5 = 0.5 * 2^{0} ==> Exp = 80, M = 00 00 00
-4 = -0.5 * 2^{3} ==> Exp = 83, M = 80 00 00
-0.25 = -0,5 * 2^{-1} ==> Exp = 7F, M = 80 00 00

Die größte Zahl, die mit all ihren Stellen in einfacher Genauigkeit dargestellt werden kann, ist $2^{24}-1$ oder 8388607 (dez.) oder 7FFFFF (hex.). Bei doppelter Genauigkeit mit der größeren Mantisse beträgt die größte genau darstellbare Zahl $2^{56}-1$ oder $3{,}578*10^{16}$ (dez.) oder 7FFFFFFFFFFFFF (hex).

Diese Werte, 8388607 oder $3{,}578*10^{16}$, stellen jedoch nicht den größten Wert dar, mit dem gerechnet werden kann, sondern nur den größten, der ohne Verlust an Genauigkeit darstellbar ist. Das liegt daran, daß der Exponent für beide Typen Werte zwischen 2^{-128} und 2^{127} annehmen kann. Theoretisch kann also das Komma (in binärer Darstellung) 127 Stellen nach rechts oder 128 Stellen nach links geschoben werden, obwohl nur 24 bzw. 56 Bits in der Mantisse stehen.

Abhängig von der Art der Daten und der Berechnung ist das auch meist ausreichend. Entscheidend ist allein die Anzahl der gültigen Ziffern eines Wertes.

Anmerkung:
Beim LASER 110, 210, 310 und dem VZ200 läßt sich nicht so ohne weiteres mit Variablen doppelter Genauigkeit arbeiten. In der Grundversion des BASIC-Interpreters können Sie nur zwischen Variablen einfacher Genauigkeit und ganzzahligen Variablen (Integer) wählen.

- Variablennamen ohne Zusatz = einfache Genauigkeit -
- Variablennamen mit Zusatz '%' = ganzzahlige Variable -

Es gibt jedoch einen kleinen Trick, indem Sie sich durch einen POKE xxxx,8 in der Typtabelle des Kommunikationsbereichs (ab 7901H) ganz bestimmte Variablennamen zu doppelter Genauigkeit definieren.

Ein-/Ausgabetreiber

Treiber haben die Aufgabe, eine Schnittstelle zwischen dem logischen Anliegen einer Anwendung (eines Programms) und den physikalischen Gegebenheiten eines speziellen Ein-/Ausgabegerätes herzustellen.

Dazu ein Beispiel:
Ein Programm möchte ein ganz bestimmtes Zeichen, z.B. den Buchstaben 'A', auf die Kassette ausgeben.
Dieses Zeichen wird im ASCII-Code im A-Register des Z80 bereitgestellt und der Treiber aufgerufen.
Der Treiber übernimmt das Zeichen und überträgt es Bit für Bit seriell in der durch das Aufzeichnungsverfahren festgelegten Pulsfolge auf den Kassettenausgang (Bit 1,2 der Adresse 6800H).

Im LASER-Computer sind Treiber für die Tastatur, den Bildschirm, einen parallelen Drucker und die Kassette vorhanden.

Tastatur- und Druckertreiber werden über einen speziellen Geräte-Steuerblock (Device Control Block = DCB) bei 7815H bzw. 7825H angesprochen. Für den Bildschirm existiert auch noch ein Rumpf-DCB bei 781DH, wird aber vom LASER, außer der Cursor-Verwaltung, nicht genutzt.

In den DCBs werden Zähler und Zeiger für das spezielle Gerät gehalten, so z.B. Papiermaß und Zeilenzähler beim Drucker oder die Cursorposition für den Bildschirm. Ferner sind dort die Treiberadressen zu finden.

Die DCBs werden bei der Systeminitialisierung im Kommunikationsbereich eingerichtet und mit Standardwerten gefüllt.

Ein Treiber wird für jedes zu übertragende Zeichen aufgerufen. Treiber kennen keine Datensätze oder Dateien, sie können nicht Zeichen blocken oder entblocken. Solche Funktionen sind vom aufrufenden Programm selbst durchzuführen. Im BASIC-Interpreter machen das z.B. die Routinen für PRINT und INPUT.

Beim Schreiben auf die Kassette erzeugt das PRINT-Kommando zunächst einen Vorspann von 255 * 80H und 5 * FEH, den Datenkenner F2H und den Dateinamen. Danach wird jede Variable als ASCII-String übertragen. Kommas trennen die einzelnen Variablen und ein Wagenrücklauf-Zeichen (CR) schließt die Übertragung ab.

Ein INPUT-Kommando sucht zuerst den Vorspann, prüft Datenkenner und Dateinamen. Alle Variablen werden danach hintereinander in den Ein-/Ausgabepuffer übertragen, bis ein Wagenrücklauf-Zeichen (CR = 0DH) entdeckt wird.

Anschließend wird jede einzelne Variable wieder in das korrekte Format umgewandelt und in die Variablentabelle übertragen.

Der Tastatur-Treiber beginnt bei Adresse 2EF4H und erstreckt sich bis zur Adresse 3014H, einschließlich einer ECHO-Routine, die die eingegebenen Zeichen direkt auf dem Bildschirm darstellt. Hinzu kommt noch eine Routine ab 0507H, die die Betätigung mehrerer Tasten gleichzeitig, das sogenannte 'ROLLOVER', behandelt.

Der Tastatur-Treiber kann direkt über den Geräte-Steuerblock (DCB) mit CALL 2BH aufgerufen werden, wenn ein einzelnes Zeichen von der Tastatur abgeholt werden soll. Das macht z.B. die Routine des BASIC-Befehls INKEY$. Bei dieser Art der Abfrage wird allerdings keine ECHO-Ausgabe auf den Bildschirm durchgeführt.

Integriert ist die Tastaturabfrage jedoch auch in der Interrupt-Service-Routine (siehe Bildschirm-Treiber Beschreibung). Diese Abfrageart wird vor allem dann benutzt, wenn eine ganze Zeile über den Bildschirm-Editor bei 3E3H eingelesen werden und der eingegebene Text auf dem Bildschirm geechod werden soll.

Der Bildschirm-Treiber erstreckt sich von 3039H bis 342FH und besteht aus verschiedenen Unterroutinen.

Hierbei ist eine besondere Problematik zu beachten. Auf den VIDEO-RAM wird von zwei verschiedenen Bausteinen zugegriffen. Zum einen tastet der Bildgenerator ständig den VIDEO-RAM ab und überträgt die dort enthaltenen Zeichen auf den Bildschirm, zum anderen muß vom Z80 die darzustellende Information in den VIDEO-RAM geschrieben werden. Geschieht dies unsynchronisiert nebeneinander, so erhalten Sie einen unschönen flackernden Bildschirm.

Dem wird abgeholfen, indem Ausgaben auf den VIDEO-RAM nur in der Dunkelphase des Bildschirms erfolgen. Gesteuert und synchronisiert wird dieser Vorgang über das vertikale Synchronisations-Signal des Bildgenerators. Dieses wird dazu benutzt, einen Interrupt zu erzeugen. Bildschirmausgaben werden zunächst in einen speziellen Puffer ab 7AAFH zwischengespeichert und erst in der Interrupt-Service-Routine, d.h. während der Dunkelphase des Bildschirms ins VIDEO-RAM übertragen.

Die Interrupt-Service-Routine befindet sich bei 2EB8H. Sie führt neben der zuvor beschriebenen gepufferten Bildschirmausgabe (über 3F7BH zu 30E8H) zusätzlich noch

- die blinkende Cursordarstellung (2EDCH),
- die Tastaturabfrage (2EFDH) und
- die ECHO-Ausgabe auf den Bildschirm (301BH) mit einem Piep-Ton (3430H)

aus.

Der Interrupt erfolgt beim PAL-System alle 20 Millisekunden. Vor Ausführung der o.a. Funktionen wird über den RAM-Erweiterungsausgang 787DH gesprungen, eine gute Gelegenheit, selbst mit dem Interrupt zu arbeiten, z.B. um eine Softwareuhr zu implementieren.

Zur Verwaltung des Bildschirms befindet sich am Ende des BASIC-Kommunikationsbereiches (ab Adresse 7AD7H) eine Tabelle, die in 16 1-Byte Einträgen Auskunft über den Status jeder einzelnen Zeile gibt.

80H - Zeile ist eine Einzelzeile von 32 Zeichen
81H - Zeile ist erste einer Doppelzeile von 64 Zeichen
00H - Zeile ist zweite einer Doppelzeile

Diese Tabelle in Verbindung mit zwei Status-Flags bei 7838H und 7839H sind Grundlage des Bildschirm-Editors bei 03E3H.

Der Drucker-Treiber beginnt bei Adresse 058DH und setzt sich bei 3AB6H fort. Dies ist der Ablauf, wenn ein normales ASCII-Zeichen ausgegeben werden soll und der Treiber über den DCB aufgerufen wird.

Eine besondere Art der Ausgabe ist der COPY-Befehl (ab 3912H), mit dessen Hilfe der Bildschirminhalt komplett ausgedruckt wird. Haben Sie einen Drucker der Marke "Seikosha GP100", oder ähnlich, angeschlossen, so können auch invertierte Zeichen (Umsetztabelle ab 39B4H), Blockgrafik und sogar Bilder der hochauflösenden Grafik ausgedruckt werden, wobei die Farben durch unterschiedliche Grautöne ersetzt werden.

Ein 'Seikosha GP100' kompatibler Drucker muß dazu folgende Bedingungen erfüllen:

- Umschaltcode Text => Grafik = 08H
- Umschaltcode Grafik => Text = 0FH
- Einzelnadelansteuerung für 7 Nadeln

Der Kassetten-Treiber erstreckt sich von 34A9H bis 389CH und besteht aus vielen einzelnen Routinen, die abhängig davon, ob eine Kasseteneingabe oder -ausgabe erfolgt, aufgerufen werden.

Die Kassettenaufzeichnungen haben folgendes Format:

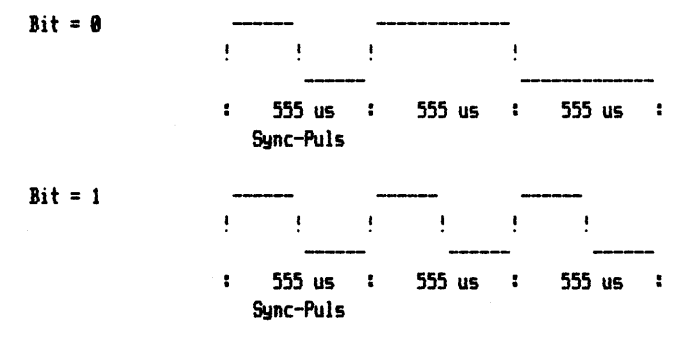
1. BASIC- und Maschinenprogramme

255 x 80H	-	Synchronisations-Bytes
5 x FEH	-	" "
1 Byte	-	Programmkennung 00 = BASIC-Programm 01 = Maschinenprogramm
max 15 Byte	-	Programmname
00H	-	Endekennung für Namensfeld
2 Byte	-	Programm-Startadresse
2 Byte	-	Programm-Endeadresse + 1
n Byte	-	Programmtext
2 Byte	-	Prüfsumme
20 x 00H	-	Endekennung

2. Daten-Dateien (aus BASIC-Programmen mit PRINT#)

255 x 80H	-	Synchronisations-Bytes
5 x FEH	-	" "
02H	-	Kennung für eine Daten-Datei
max 15 Byte	-	Dateiname
00H	-	Endekennung für Namensfeld
n Byte	-	String-Variable, durch Komma getrennt
0DH	-	Endekennung (Carriage Return)

Die Kassettenaufzeichnung erfolgt bitseriell in der folgenden Signalform:



Zunächst wird ein Synchronisationspuls von 555 Mikrosekunden Dauer ausgegeben. Bei einer binären Null folgt ein langer Puls von insgesamt 1110 Mikrosekunden, bei einer binären Eins folgen zwei Pulse von je 555 Mikrosekunden. Das ergibt eine Gesamtpulslänge für eine Bitaufzeichnung von 1665 Mikrosekunden und entspricht einer Aufzeichnungsrate von 600 Bit/Sek. (Baud).

Die von der Kassetten-Treiberroutine erzeugten Meldungen in der letzten Bildschirmzeile können bei Bedarf unterdrückt werden. Sie können dies über die Adresse 784CH bewerkstelligen.

784CH = 0 Meldungen werden ausgegeben
784CH > 0 Meldungen werden unterdrückt.



9. Adressen und Tabellen des BASIC-Interpreters

```
0000H =>  -------------------
          !     Internes    !
          !  Betriebssystem !
          !      (ROM)      !
4000H =>  -------------------
          !    frei oder    !
          ! belegt durch DOS !
6000H =>  -------------------
          !      frei       !
6800H =>  -------------------
          ! Ein-/Ausgabebereich!
7000H =>  -------------------
          !   Bildschirm-   !
          !     speicher    !
7800H =>  -------------------
          !      BASIC      !
          ! Kommunikationsber. !
(78A4H)=> -------------------
          !  Programmtabelle !
          !      (PST)      !
(78F9H)=> -------------------
          !Variablentabelle-VLT!
          !  einfache Variable !
(78FBH)=> -------------------
          !indizierte Variable !
(78FDH)=> -------------------
          !                 !
          !   Freispeicher  !
          !                 !
(78E8H)=> -------------------
          !   BASIC - Stack  !
(78A0H)=> -------------------
          !  String - Bereich !
(78B1H)=> -------------------
```

nnnn = absolute Lage des Bereichs im Speicher
(nnnn) = Adreßzeiger im Kommunikationsbereich

Interne Tabellen

Als 'Interne Tabellen' werden die Listen und Tabellen bezeichnet, die sich fest im ROM des Betriebssystems befinden. Das bedeutet gleichzeitig, daß Inhalt und Lage dieser Tabellen fest sind. Sie werden vom BASIC-Interpreter zur syntaktischen Prüfung, zur Ausdrucksanalyse, zur Datenumwandlung und zur Ausführung bestimmter Befehle (z.B. FOR, IF) benutzt.

BASIC-Schlüsselworttabelle (1650H - 1821H)

Diese Tabelle enthält alle reservierten Worte und Zeichen des BASIC-Interpreters, seien es Anweisungen oder Funktionen.

Jeder Eintrag enthält ein Wort, wobei im ersten Zeichen des Eintrags das Bit $2^7 = 1$ ist.

Während der Eingabe-Phase wird eine eingelesene Zeile Zeichen für Zeichen mit dieser Tabelle verglichen. Stimmt ein Textabschnitt der Zeile mit einem Eintrag überein, so wird er durch das sogenannte TOKEN ersetzt. Ein TOKEN ist ein 1-Byte hexadezimaler Wert, der aus der lfd. Eintragsnummer des gefundenen Eintrags gebildet wird. Zusätzlich wird Bit 2^7 gesetzt.

Beispiel:
CLS ist der 5. Eintrag in der Tabelle.
Da mit 0 begonnen wird zu zählen, ergibt das eine Eintragsnummer von 4.
Wird Bit 2^7 eingeschaltet, so ergibt sich ein TOKEN von 84H für CLS.
D.h. jede Zeichenfolge CLS im Programmtext wird durch 84H ersetzt, wobei die Zeilenlänge entsprechend um 2 Byte verkürzt wird.

Wenn Sie die Tabelle untersuchen, so werden Sie feststellen, daß eine ganze Reihe von Schlüsselworten nicht kodiert ist, sondern aus binären Nullen besteht. Diese sind in der nachstehend aufgeführten Liste in Klammern eingeschlossen und mit einem Stern gekennzeichnet. Soweit es sich nicht um DOS-Befehle handelt, sind jedoch die Ausführungsroutinen größtenteils vorhanden.

M.E. ist der Grund im Urheberrecht zu suchen, um eine zu große Ähnlichkeit mit einem anderen Rechner zu vermeiden. Mit EXTENDED BASIC werden viele dieser Befehle wieder für den Nutzer verfügbar gemacht.

Schlüsselwort	TOKEN	Schlüsselwort	TOKEN	Schlüsselwort	TOKEN
END	80H	(NAME)	A9H *	OR	D3H
FOR	81H	(KILL)	AAH *	>	D4H
RESET	82H	(LSET)	ABH *	=	D5H
SET	83H	(RSET)	ACH *	<	D6H
CLS	84H	(SAVE)	ADH *	SGN	D7H
(CMD)	85H *	(SYSTEM)	AEH *	INT	D8H
(RANDOM)	86H *	LPRINT	AFH	ABS	D9H
NEXT	87H	(DEF)	B0H *	(FRE)	DAH *
DATA	88H	POKE	B1H	INP	DBH
INPUT	89H	PRINT	B2H	(POS)	DCH *
DIM	8AH	CONT	B3H	SQR	DCH
READ	8BH	LIST	B4H	RND	DEH
LET	8CH	LLIST	B5H	LOG	DFH
GOTO	8DH	(DELETE)	B6H *	EXP	E0H
RUN	8EH	(AUTO)	B7H *	COS	E1H
IF	8FH	CLEAR	B8H	SIN	E2H
RESTORE	90H	CLOAD	B9H	TAN	E3H
GOSUB	91H	CSAVE	BAH	ATN	E4H
RETURN	92H	NEW	BBH	PEEK	E5H
REM	93H	TAB(	BCH	(CVI)	E6H *
STOP	94H	TO	BDH	(CVS)	E7H *
ELSE	95H	(FN)	BEH *	(CVD)	E8H *
COPY	96H	USING	BFH	(EOF)	E9H *
COLOR	97H	(VARPTR)	C0H *	(LOC)	EAH *
VERIFY	98H	USR	C1H	(LOF)	EBH *
(DEFINT)	99H *	(ERL)	C2H *	(MKI$)	ECH *
(DEFSNG)	9AH *	(ERR)	C3H *	(MKS$)	EDH *
(DEFDBL)	9BH *	(STRING$)	C4H *	(MKD$)	EEH *
CRUN	9CH	(INSTR)	C5H *	(CINT)	EFH *
MODE	9DH	POINT	C6H	(CSNG)	F0H *
SOUND	9EH	(TIME$)	C7H *	(CDBL)	F1H *
(RESUME)	9FH *	(MEM)	C8H *	(FIX)	F2H *
OUT	A0H	INKEY$	C9H	LEN	F3H
(ON)	A1H *	THEN	CAH	STR$	F4H
(OPEN)	A2H *	NOT	CBH	VAL	F5H
(FIELD)	A3H *	STEP	CCH	ASC	F6H
(GET)	A4H *	+	CDH	CHR$	F7H
(PUT)	A5H *	-	CEH	LEFT$	F8H
(CLOSE)	A6H *	*	CFH	RIGHT$	F9H
(LOAD)	A7H *	/	D0H	MID$	FAH
				'	FBH

Adreßtabellen der Ausführungsroutinen

Im ROM existieren zwei verschiedene Adreßtabellen für die Ausführungsroutinen. Die erste wird von der Ausführungssteuerung benutzt, wenn eine BASIC-Anweisung auszuführen ist. Sie enthält Adressen für die Ausführungsroutinen der TOKEN 80H bis BBH und befindet sich im Adreßbereich 1822H - 1899H. Das erste TOKEN einer Anweisung (Bit7=0) wird als Index (0 - 59) für den Tabellenzugriff benutzt. Aus der Tabelle wird die Adresse der Ausführungsroutine entnommen und diese aufgerufen. Beginnt die Anweisung nicht mit einem TOKEN, so wird zur Routine für Wertzuweisungen verzweigt (implizites LET).

Die zweite Adreßtabelle von 1608H - 164FH enhält Adressen von Routinen für BASIC-Funktionen, die nur auf der rechten Seite einer Wertzuweisung (nach dem Gleichheitszeichen) auftreten dürfen.

Wird während der Ausdrucksanalyse ein TOKEN im Bereich D7H bis FAH angetroffen, so wird dieser als Index (0 - 35) für die zweite Adreßtabelle benutzt und dort die Adresse der Ausführungsroutine entnommen.

Für die TOKEN BCH bis D6H wird keine Adreßtabelle benötigt, da diese bei Auftreten direkt von den anderen Ausführungsroutinen abgearbeitet werden.

Adreßtabelle der BASIC-Anweisungen (1822H - 1899H)

TOKEN	Schlüsselwort	Adresse
80	END	1DAE
81	FOR	1CA1
82	RESET	0138
83	SET	0135
84	CLS	01C9
85	CMD	7973
86	RANDOM	01D3
87	NEXT	22B6
88	DATA	1F05
89	INPUT	219A
8A	DIM	2608
8B	READ	21EF
8C	LET	1F21
8D	GOTO	1EC2
8E	RUN	1EA3
9E	SOUND	2BF5
9F	RESUME	1FAF
A0	OUT	2AFB
A1	ON	1F6C
A2	OPEN	7979
A3	FIELD	797C
A4	GET	797F
A5	PUT	7982
A6	CLOSE	7985
A7	LOAD	7988
A8	MERGE	798B
A9	NAME	798E
AA	KILL	7991
AB	LSET	7997
AC	RSET	799A

TOKEN	Schlüsselwort	Adresse	TOKEN	Schlüsselwort	Adresse
8F	IF	2039	AD	SAVE	79A0
90	RESTORE	1D91	AE	SYSTEM	0000
91	GOSUB	1EB1	AF	LPRINT	2067
92	RETURN	1EDE	B0	DEF	795B
93	REM	1F07	B1	POKE	2CB1
94	STOP	1DA9	B2	PRINT	206F
95	ELSE	1F07	B3	CONT	1DE4
96	COPY	3912	B4	LIST	2B2E
97	COLOR	389D	B5	LLIST	2B29
98	VERIFY	3738	B6	DELETE	2BC6
99	DEFINT	1E03	B7	AUTO	2008
9A	DEFSNG	1E06	B8	CLEAR	1E7A
9B	DEFDBL	1E09	B9	CLOAD	3656
9C	CRUN	372E	BA	CSAVE	34A9
9D	MODE	2E63	BB	NEW	1B49

Adreßtabelle der BASIC-Funktionen (1608 - 164F)

TOKEN	Schlüsselwort	Adresse	TOKEN	Schlüsselwort	Adresse
D7	SGN	098A	E9	EOF	7961
D8	INT	0B37	EA	LOC	7964
D9	ABS	0977	EB	LOF	7967
DA	FRE	27D4	EC	MKI$	796A
DB	INP	2AEF	ED	MKS$	796D
DC	POS	27F5	EE	MKD$	7970
DD	SQR	13E7	EF	CINT	0A7F
DE	RND	14C9	F0	CSNG	0AB1
DF	LOG	0809	F1	CDBL	0ADB
E0	EXP	1439	F2	FIX	0B26
E1	COS	1541	F3	LEN	2A03
E2	SIN	1547	F4	STR$	2836
E3	TAN	15A8	F5	VAL	2AC5
E4	ATN	15BD	F6	ASC	2A0F
E5	PEEK	2CAA	F7	CHR$	2A1F
E6	CVI	7952	F8	LEFT$	2A61
E7	CVS	7958	F9	RIGHT$	2A91
E8	CVD	795E	FA	MID$	2A9A

Rangfolge der Rechenoperationen

Die Rangfolge der verschiedenen Rechenoperationen in arithmetischen Ausdrücken wird ebenfalls mit Hilfe einer Tabelle ermittelt, die sich im Adreßbereich 189AH bis 18A0H befindet.

Diese Tabelle enthält für die verschiedenen Operatoren numerische Werte, mit denen die Rangfolge festgelegt ist.

Bei der Ausdrucksanalyse wird jedes Operator/Operanden-Paar plus dem Rang-Wert des vorausgegangenen Operators auf den Stack gelegt. Wird ein Operator mit höherem Rangwert als der vorausgegangene gefunden, so wird die Operation sofort ausgeführt und das damit erhaltene Zwischenergebnis auf den Stack gepackt.

Operator	Funktion	Rang-Wert
+	Addition	79
-	Subtraktion	79
*	Multiplikation	7C
/	Division	7C
[	Potenzieren	7F
AND	logische Verknüpfung	50
OR	logische Verknüpfung	46

Arithmetische Routinen

Für die drei unterschiedlichen Typen numerischer Variablen enthält das ROM bei 18ABH-18C8H drei Tabellen mit den Anfangsadressen der zugehörigen arithmetischen Routinen. Diese werden von der Ausdrucksanalyse benutzt.

Funktion	Ganzzahlige Variable	Variable einf.Genauigk.	Variable dopp.Genauigk.
Addition	0BD2	0716	0C77
Subtraktion	0BC7	0713	0C70
Multiplikation	0BF2	0847	0DA1
Division	2490	08A2	0DE5
Vergleich	0A39	0A0C	0A78

Zur Vollständigkeit noch die Adresse der Routine zur Addition von Textvariablen (Strings). Diese liegt bei 298FH.

Daten - Umwandlung (Typanpassung)

Zur Umwandlung von Daten in die verschiedenen Variablentypen existiert eine weitere Tabelle, die, abhängig vom Zieltyp, die Adressen der entsprechenden Umwandlungsroutinen enthält. Diese Routinen wandeln den im Arbeitsbereich 1 (X-Register) befindlichen Wert in den gewünschten Datentyp um. Benutzt werden sie vornehmlich von der Ausdrucksanalyse um Werte und Zwischenergebnisse verschiedenen Typs miteinander verknüpfen zu können.

Die Tabelle befindet sich bei 18A1H-18AAH.

Umwandlung in	Adresse
Textvariable	0AF4 *
Ganzzahlige Variable	0A7F
Variable einf.Genauigkeit	0AB1
Variable dopp.Genauigkeit	0ADB

* Für Textvariable enthält die angegebene Routine keine Umwandlung, sondern nur einen Test, ob die im Arbeitsbereich 1 befindliche Variable auch wirklich eine Textvariable ist. Wenn nicht, wird 'TYPE MISMATCH ERROR' ausgegeben.

Fehlermeldungen

Im ROM-Bereich befinden sich zwei Tabellen mit Fehlermeldungen.

Die erste Tabelle befindet sich bei 18C9H-18F6H und enthält pro Fehlermeldung nur eine zwei Zeichen lange Abkürzung. Sie ist ein Relikt aus einem anderen Einsatzrechner dieses Betriebssystems und wird vom LASER-Rechner nicht benutzt.

Die von den LASER 110, 210, 310 und dem VZ200 benutzte Tabelle liegt bei 3CECH-3E28H und enthält die Fehlermeldungen in ausgeschriebener Form.

Die Fehlermeldungen werden anhand einer Fehlernummer ermittelt, die als Index für den Tabellenzugriff genutzt wird.

Fehlernummer	Fehlercode	Fehlermeldung (lang)
0	NF	NEXT WITHOUT FOR
2	SN	SYNTAX ERROR
4	RG	RET'N WITHOUT GOSUB
6	OD	OUT OF DATA
8	FC	FUNCTION CODE ERROR
A	OV	OVERFLOW
C	OM	OUT OF MEMORY
E	UL	UNDEF'D STATEMENT
10	BS	BAD SUBSCRIPT
12	DD	REDIM'D ARRAY
14	0/	DIVISION BY ZERO
16	ID	ILLEGAL DIRECT
18	TM	TYPE MISMATCH
1A	OS	OUT OF SPACE
1C	LS	STRING TOO LONG
1E	ST	FORMULA TOO COMPLEX
20	CN	CAN'T CONTINUE
22	NR	NO RESUME
24	RW	RESUME WITHOUT ERROR
26	UE	UNPRINTABLE ERROR
28	MO	MISSING OPERAND
2A	FD	BAD FILE DATA
2C	L3	DISK COMMAND ERROR

Externe Tabellen

Als 'Externe Tabellen' werden die Datenstrukturen bezeichnet, die vom BASIC-Interpreter im RAM-Bereich angelegt und verwaltet werden. Merkmal der externen Tabellen ist, daß sich ihr Inhalt ändern kann und auch teilweise ihre Lage im RAM-Bereich.

Für Tabellen, die ihre Lage im Speicher verändern, befinden sich im BASIC-Kommunikationsbereich Adreßzeiger, so daß sie jederzeit wieder aufgefunden und adressiert werden können.

Der BASIC - Kommunikationsbereich (7800H - 7AE8H)

Im Kommunikationsbereich werden vom BASIC-Interpreter alle erforderlichen Adreßzeiger und Verwaltungstabellen angelegt und verwaltet, die sich während der normalen Programmbearbeitung ändern können bzw. in denen der Nutzer ggf. auch eigene Ablaufvariable definieren oder ändern kann.

Betrachten Sie den Kommunikationsbereich als den Notizblock des BASIC-Interpreters.

Die nachfolgende Auflistung enthält eine Beschreibung jedes einzelnen Bytes in diesem Bereich.

Einige Tabellen innerhalb des Kommunikationsbereichs werden im Anschluß an diese Auflistung noch ausführlich beschrieben.

Der Bereich 7800H bis 7835H wird bei der System-Initialisierung aus dem ROM-Bereich vorbelegt.

```
7800   C3 96 1C    JP    1C96H        ;RST 8 - Vektor
7803   C3 78 1D    JP    1D78H        ;RST 10 - Vektor
7806   C3 90 1C    JP    1C90H        ;RST 18 - Vektor
7809   C3 D9 25    JP    25D9H        ;RST 20 - Vektor
780C   C9 00 00    RET                ;RST 28 - Vektor
780F   C9 00 00    RET                ;RST 30 - Vektor
7812   FB          EI                 ;RST 38 - Vektor
7813   C9 00       RET
```

```
                        Tastatur Device-Control-Block (DCB)
7815    01              DCB-Kenner
7816    F4 2E           Treiber-Adresse
7818    00              Hintergrund-Flag
                        (0=grün, 1=schwarz)
7819    00              akt. Hintergrund
781A    00
781B    4B 49           'KI'

                        Bildschirm Device-Control-Block (DCB)
                        (im LASER 110-310 unbenutzt)
781D    00              DCB-Kenner (gelöscht)
781E    00 00           Zeiger auf Programm-Anfangsadresse
                        bei CLOAD
7820    00 70           Cursor-Adresse
7822    00
7823    00 00           Prüfsumme bei Kassetten Ein-/Ausgabe

                        Drucker Device-Control-Block (DCB)
7825    06              DCB-Kenner
7826    8D 05           Treiber-Adresse
7828    43              Zeilen/Seite+1
7829    00              Zeilenzähler
782A    00
782B    50 52           'PR'

782D    C3 00 50        JP     5000H          ;unbenutzt
7830    C7 00 00        RST    0              ;unbenutzt

7833    3E 00           LD     A,0            ;bei unbekannter DCB-Kennung A=0
7835    C9              RET

7836                    Puffer B1 für 1. Tastencode bei gleichzeitig mehrfacher
                        Tastenbetätigung
7837                    Puffer B2 für 2. Tastencode bei gleichzeitig mehrfacher
                        Tastenbetätigung
```

7838	FLAG 1 Bit 7 - CONTROL-Flag Bit 6 - REPEAT-Flag Bit 5 - WAIT-Flag Bit 4 - B2-Status-Flag Bit 3 - B1-Status-Flag Bit 2 - FUNCTION-Flag Bit 1 - INVERSE-Flag Bit 0 - SHIFT-Flag
7839	FLAG 2 Bit 7 - unbenutzt Bit 6 - CRUN-Flag Bit 5 - Ini-Flag f. gepufferte Ausgabe Bit 4 - Flag f. INPUT-Anweisung Bit 3 - VERIFY-Flag Bit 2 - BREAK-Flag Bit 1 - BUZZER-Flag Bit 0 - Carriage-Return Flag
783A	Zeitzähler
783B	INPUT/OUTPUT-Latch
783C	Zeichensicherung für Cursor-Darstellung
783D-7840	unbenutzt
7841	Blink-Zähler
7842-7843	Zwischenspeicher bei Tastaturabfrage (Zeile/Spalte)
7844-7845	Zwischenspeicher bei Tastaturabfrage (Matrix-Adresse)
7846	Farb-Code
7847-784B	unbenutzt
784C	Ausgabe-Flag f. Meldungsausgabe bei Kassetten I/O (>0 - Meldungen werden unterdrückt)
784D-787C	unbenutzt

```
7870    C9 00 00    RAM-Erweiterungsausgang der Interrupt-Service-Routine

                    Der Bereich 7880H-78A5H wird bei der Initialisierung aus
                    dem ROM-Bereich gefüllt

                    Unterprogramm für Division
7880    D6 00       SUB   0             ;Subtraktion Z2 - Z1
7882    6F          LD    L,A           ;wird vor jedem Aufruf modifiziert.
7883    7C          LD    A,H
7884    DE 00       SBC   A,0
7886    67          LD    H,A
7887    78          LD    A,B
7888    DE 00       SBC   A,0
788A    47          LD    B,A
788B    3E 00       LD    A,0
788D    C9          RET

788E    4A 1E       USR - Startadresse
                    vorbesetzt mit FUNCTION-CODE Error

7890    40 E6 4D    Multiplikator f. RND

                    Unterprogramm für INP
7893    DB 00       IN    A,(0)
7895    C9          RET

                    Unterprogramm für OUT
7896    D3 00       OUT   (0),A

7899    00          INKEY$-Zwischenspeicher

789A    00          letzter Fehlercode für ERR

789B    00          Druckerposition in der Zeile

789C    00          Geräte-Flag (0=Bildsch., 1=Drucker, 80=Kassette)

789D    40          Zeilenlänge auf dem Bildschirm (vorbesetzt mit 64)

789E    30          letzte Tabulator-Position (vorbesetzt mit 48)

789F    00          unbenutzt

78A0    47 7B       Anfangsadresse des String-Bereichs
```

78A2	FE FF	aktuelle Zeilennummer
78A4	E9 7A	Anfangsadresse des Programmtextes
78A6-78A7		Spaltenzeiger für Ausgabebild
78A7-78A8		Zeiger auf Ein-/Ausgabe-Puffer (ab 79E8H)
78A9		Eingabe-Flag (0 = Kassette)
78AA-78AD		letzte Zufallszahl
78AE		Flag für DIM-Anweisung
78AF		Typ des Wertes im X-Register 02 = Integer 03 = String 04 = einfache Genauigkeit 08 = doppelte Genauigkeit
78B0		Flag für Zwischencode-Erzeugung bei DATA Operationscode bei der Ausdrucksanalyse
78B1-78B2		Endadresse des BASIC-Speicherbereichs
78B3-78B4		Zeiger auf String-Zwischenspeicher
78B5-78D2		String-Zwischenspeicher (10 x 3 Bytes) (1 Byte - Länge, 2 Byte - Adresse im Stringbereich)
78D3-78D5		vorl. String-Zwischenspeicher
78D6-78D7		Zeiger auf letztes freies Byte im Stringbereich
78D8-78D9		Allg. Adresszwischenspeicher Formatflag f. Stringausgabe einer Zahl
78DA-78DB		DATA - Zeilennummer
78DC		Flag zur Sperrung der Indizierung
78DD		RESUME/RETURN - Flag

78DE	Zwischenpuffer für PRINT USING DATA-Flag für INPUT u.a.
78DF-78E0	allgemeiner Adress-Speicher z.B. Programmfortführung bei NEW Laufvariable bei FOR/NEXT Adr. d. Variablentabelle bei LET
78E1	AUTO-Eingabe - Flag (0 - kein AUTO)
78E2-78E3	AUTO - Zeilennummer
78E4-78E5	AUTO - Erhöhungswert
78E6-78E7	Adresse der aktuellen Zeile (FFFF = Direktkommando)
78E8-78E9	Zeiger auf den BASIC-Stack
78EA-78EB	Nummer der Zeile, in der der letzte Fehler auftrat
78EC-78ED	Nummer der Zeile, in der der letzte Fehler auftrat (.-Option bei LIST)
78EE-78EF	Adresse der Zeile, in der der Fehler auftrat
78F0-78F1	Adresse einer Fehlerbehandlungs-Routine (ON ERROR)
78F2	Fehler - Flag (Fehler=255, RESUME=0)
78F3-78F4	Adresse des Dezimal-Punktes im Druck-Puffer
78F5-78F6	Zeilennummer, bei der die letzte Unterbrechung stattfand (END, STOP, BREAK)
78F7-78F8	Adresse der Zeile, in der die letzte Unterbrechung stattfand
78F9-78FA	Programm-Endadresse Anfang der Variablen-Tabelle
78FB-78FC	Endadresse des 1. Teils der Variablen-Tabelle Anfang der Matrix-Tabelle (2. Teil)

78FD-78FE	Anfangsadresse des freien Speichers (hinter der Matrix-Tabelle)
78FF-7900	Zeiger auf DATA-Zeile
	<u>Typcode - Tabelle</u>
7901	A
7902	B
7903	C
7904	D
7905	E
7906	F
7907	G
7908	H
7909	I
790A	J
790B	K
790C	L
790D	M
790E	N
790F	O
7910	P
7911	Q
7912	R
7913	S
7914	T
7915	U
7916	V
7917	W
7918	X
7919	Y
791A	Z
791B	TRACE -FLAG (0 = TRON, AF = TROFF)

X - Register

7910 zus. Byte für rechts schieben

	INT	STRING	SINGLE	DOUBLE
791D				LSB
791E				LSB
791F				LSB
7920				LSB
7921	LSB	ADR	LSB	LSB
7922	MSB	ADR	LSB	LSB
7923			MSB	MSB
7924			EXP	EXP

7925 Zwischenspeicher für Arithmetik-Operationen. z.B. Vorzeichen

7926-792E Y - Register (Aufteilung wie X-Register)

792F unbenutzt

7930-7949 Druck-Puffer

794A-7951 Zusätzliches Register für Multiplikationen und Divisionen mit doppelter Genauigkeit

RAM-Vektoren für Disketten-Befehle
vorbesetzt mit 'JP 012DH' (DISK-COMMAND - Error)

7952 CVI-Anweisung
7955 FN - Anweisung
7958 CVS-Anweisung
795B DEF-Anweisung
795E CVD-Anweisung
7961 EOF-Anweisung
7964 LOC-Anweisung
7967 LOF-Anweisung
796A MKI$-Anweisung
796D MKS$-Anweisung
7970 MKD$-Anweisung
7973 CMD-Anweisung
7976 TIME$-Anweisung
7979 OPEN-Anweisung

797C	FIELD-Anweisung
797F	GET-Anweisung
7982	PUT-Anweisung
7985	CLOSE-Anweisung
7988	LOAD-Anweisung
798B	MERGE-Anweisung
798E	NAME-Anweisung
7991	KILL-Anweisung
7994	& - Anweisung
7997	LSET-Anweisung
799A	RSET-Anweisung
799D	INSTR-Anweisung
79A0	SAVE-Anweisung
79A3	LINE-Anweisung

<u>RAM-Erweiterungsausgänge</u>
vorbelegt mit C9H-00H-00H (RET)

79A6	aus ERROR-Routine
79A9	aus USR-Routine
79AC	Anfang BASIC-Schleife
79AF-79B1	unbenutzt
79B2	aus Programm-Eingabe
79B5	Ende Programmeingabe
79B8	Ende Programmeingabe
79BB	aus NEW und END
79BE	Endabfrage PRINT
79C1	Datenausgabe
79C4	Einlesen v. Tastatur
79C7	RUN-Ausführung
79CA	Anfang PRINT-Anweisung
79CD	PRINT-Anweisung
79D0	PRINT-Anweisung
79D3	PRINT-Anweisung
79D6	INPUT-Anweisung
79D9	MID$ als Anweisung
79DC	INPUT-Anweisung
79DF	READ + INPUT + LIST
79E2-79E4	unbenutzt

79E5   3A 00 2C   I/O-Buffer-Vorspann

79E8-7A9C	Ein/Ausgabe-Puffer
79F8	BASIC-Stack während der Initialisierung
7A9D-7AAD	Programm-/Dateiname - Zwischenspeicher bei Kassetten-Ein/Ausgabe
7AAE	Spaltenanzeige auf dem Bildschirm
	Zusätzlicher Ausgabepuffer für gepufferte Bildschirmausgabe
7AAF	Anzahl Zeichen im Puffer
7AB0-7AB1	Puffer-Zeiger
7AB2-7AD1	Puffer-Bereich
7AD2-7AD5	4 Byte-Puffer für Grafik-Druck, SOUND u. Kassette I/O
7AD6	Zähler f. o.a.Puffer + Länge Namen bei Kassetten I/O
	Zeilenstatus für Bildschirm-Zeilen (80=Einzelzeile, 81=Doppelzeile, 00=Folgezeile)
7AD7	Zeile 1
7AD8	Zeile 2
7AD9	Zeile 3
7ADA	Zeile 4
7ADB	Zeile 5
7ADC	Zeile 6
7ADD	Zeile 7
7ADE	Zeile 8
7ADF	Zeile 9
7AE0	Zeile 10
7AE1	Zeile 11
7AE2	Zeile 12
7AE3	Zeile 13
7AE4	Zeile 14
7AE5	Zeile 15
7AE6	Zeile 16
7AE7	unbenutzt
7AE8	Programm-Anfang

Der String-Zwischenspeicher (78B3H - 78D2H)

Dies ist eine Tabelle innerhalb des Kommunikationsbereichs. Sie wird vom BASIC-Interpreter zur Zwischenspeicherung von Text-Variablen (Strings) benutzt, die bei String-Additionen oder einigen PRINT-Operationen anfallen.

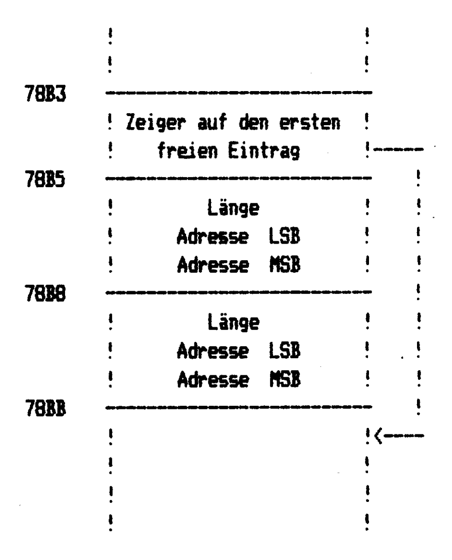
Die Tabelle besteht aus zehn 3-Byte Einträgen, die nacheinander zugewiesen werden. Am Anfang der Tabelle, bei 78B3H, befindet sich ein Adreßzeiger auf den nächsten freien Eintrag. Bei Initialisierung des Rechners wird er auf den ersten Eintrag der Tabelle gesetzt.

Jeder Eintrag besteht aus einem Längenbyte und einem Adreßzeiger auf den String im Stringbereich oder in der Programm-Tabelle. Einträge werden von oben nach unten zugewiesen und von unten nach oben wieder freigegeben.

Bei überlauf der Tabelle wird die Fehlermeldung

FORMULA TOO COMPLEX

ausgegeben.



Die Typ-Code - Tabelle (7901H - 791AH)

Diese Tabelle wird vom BASIC-Interpreter benutzt, um den Datentyp einer Variablen zu ermitteln (Ganzzahlig, String, einf. Genauigkeit, dopp. Genauigkeit).

Die Lage dieser Tabelle ist ebenfalls fest im Kommunikationsbereich, daher ist zur Adressierung auch kein Adreßzeiger erforderlich. Der Inhalt ist änderbar; bei normalem BASIC mit Hilfe des POKE-Befehls, im EXTENDED BASIC mit den DEFxxx-Deklarationen.

Die Tabelle ist 26 Byte lang, das entspricht der Anzahl Zeichen des englischen Alphabets. Für jeden Buchstaben von 'A' bis 'Z' ist ein Byte vorhanden. Als Index für den Tabellenzugriff wird der erste Buchstabe eines Variablennamens benutzt. Jeder Eintrag enthält einen Code, der Auskunft über den vereinbarten Variablentyp gibt:

02 - ganzzahlige Variable
03 - Text-Variable (String)
04 - Variable einfacher Genauigkeit
08 - Variable doppelter Genauigkeit

Bei der Systeminitialisierung werden alle Einträge mit 04, d.h. als Variable einfacher Genauigkeit gekennzeichnet.

Enthält ein Variablenname bereits eine Typen-Kennung (z.B. A$ oder B1%), so hat diese Vorrang vor einem anderslautenden Tabelleneintrag.

Adresse	Buchstabe	Typ bei Initialisierung
7901	A	04
7902	B	04
7903	C	04
7904	D	04
7905	E	04
7906	F	04
7907	G	04
7908	H	04
7909	I	04
790A	J	04
790B	K	04

790C	L	04
790D	M	04
790E	N	04
790F	O	04
7910	P	04
7911	Q	04
7912	R	04
7913	S	04
7914	T	04
7915	U	04
7916	V	04
7917	W	04
7918	X	04
7919	Y	04
791A	Z	04

Zeilenstatus - Tabelle (7AD7H - 7AE6H)

Am Ende des Kommunikationsbereichs befindet sich eine 16 Byte Tabelle, die vom Bildschirm-Editor zur Zeilenverwaltung benötigt wird. Sie enthält für jede der 16 Bildschirmzeilen einen 1-Byte Eintrag mit folgendem Inhalt:

80 - bei der Zeile handelt es sich um eine Einzelzeile von 32 Zeichen.

81 - diese Zeile ist die erste einer Doppelzeile von 64 Zeichen

00 - diese Zeile ist die zweite einer Doppelzeile von 64 Zeichen.

Programm - Tabelle (Program Statement Table = PST)

Ein eingegebenes oder von Kassette bzw. Diskette eingelesenes BASIC-Programm wird mit all seinen Anweisungszeilen in der Programmtabelle gespeichert. Diese schließt sich normalerweise direkt an den Kommunikationsbereich an.

Da sich die Programm-Tabelle im Freispeicherbereich des RAM befindet und sich ihre Lage ggf. ändern kann, befindet sich im Kommunikationsbereich bei 78A4H

ein Zeiger auf die Tabellen-Anfangsadresse und bei 78F9H ein Zeiger auf das Tabellenende+1.

Eingegebene Programmzeilen werden von der Eingabe-Routine komprimiert, d.h. BASIC-Schlüsselworte werden durch TOKEN ersetzt, und in die Programmtabelle übertragen. Die einzelnen Programmzeilen werden aufsteigend nach Zeilennummern gespeichert, unabhängig davon, in welcher Reihenfolge sie eingegeben wurden.

Jede Programmzeile beginnt mit einem Adreßzeiger auf den Anfang der nächsten Zeile. Darauf folgt die Zeilennummer im 2-Byte Integer- (ganzzahligem) Format. Hinter der Zeilennummer steht der eigentliche Zeilentext, der mit einem 'Null'-Byte (00H) beendet wird. Dieses 'Null'-Byte bezeichnet man auch als "End of Statement"-Flag oder EOS.

Das Programmende wird durch zwei weitere 'Null'-Bytes nach der letzten Programmzeile gekennzeichnet.

```
(78A4H)    -------------------------
           !  2-Byte Adreßzeiger    !----->
           -------------------------      !
           !  2-Byte Zeilennummer   !     !
           -------------------------      !
           !      Zeilentext        !     !
           !                        !     !
           -------------------------      !
           !    Zeilenende  00H     !     !
           -------------------------<-----!
           !  2-Byte Adreßzeiger    !----->
           -------------------------      !
           !  2-Byte Zeilennummer   !     !
           -------------------------      !
           !      Zeilentext        !     !
           !                        !     !
           -------------------------      !
           !    Zeilenende  00H     !     !
           -------------------------<------
           !  2-Byte Adreßzeiger    !----->
           .                        .     !
           .                        .
           -------------------------      !
           !    Zeilenende  00H     !     !
           -------------------------<------
           ! Programmende  00H-00H !
(78F9H)=>  -------------------------
```

Der Inhalt der Programm-Tabelle soll anhand eines einfachen Bespieles von zwei Programmzeilen dargestellt werden.

```
        420  IF A = 25 THEN 500
        430  A = A + 1


 Zeiger von vor-  !                        !
 heriger Zeile    ------------------------- <----  = 8132H
                  !           45           !
                  !           81           ! ---->
       420        !           A4           !     !
                  !           01           !     !
        IF        !           8F           !     !
                  !           20           !     !
        A         !           41           !     !
                  !           20           !     !
        =         !           D5           !     !
                  !           20           !     !
        2         !           32           !     !
        5         !           35           !     !
                  !           20           !     !
       THEN       !           CA           !     !
                  !           20           !     !
        5         !           35           !     !
        0         !           30           !     !
        0         !           30           !     !
                  !           00           !     !
   8145H ====>    !           53           !<-----
                  !           81           !------>
       430        !           AE           !     !
                  !           01           !     !
        A         !           41           !     !
                  !           20           !     !
        =         !           D5           !     !
                  !           20           !     !
        A         !           41           !     !
                  !           20           !     !
        +         !           CD           !     !
                  !           20           !     !
        1         !           31           !     !
                  !           00           !     !
   8153H ====>    !                        !<-----
```

Die Variablen - Tabelle

In dieser Tabelle sind alle Variablen enthalten, die in einem BASIC-Programm definiert und zugewiesen wurden. Die Tabelle ist in zwei Abschnitte unterteilt. Abschnitt 1 enthält alle einfachen Variablen, im Abschnitt 2 sind alle dimensionierten Variablen, d.h. Matrizen, verzeichnet.

Drei Adreßzeiger im Kommunikationsbereich geben Auskunft über die Lage der Variablen-Tabelle im RAM-Bereich.

```
78F9H - Anfangsadresse des 1. Abschnitts
        (einfache Variable)
78FBH - Anfangsadresse des 2. Abschnitts
        (Matrizen)
78FDH - Endadresse der Variablentabelle + 1
        (= Beginn des Freispeichers)
```

```
             !                  !
             !  Programm-Tabelle !
(78F9H) ===> --------------------
             !     einfache      !
             !     Variable      !
(78FBH) ===> --------------------
             !   dimensionierte  !
             !     Variable      !
             --------------------
(78FDH) ===> !    Freispeicher   !
             !                   !
```

Unabhängig davon, in welchem Abschnitt eine Variable definiert ist, die ersten drei Bytes jedes Eintrags haben das gleiche Format.
Byte 1 enthält den Typcode der Variablen (02, 03, 04 oder 08), der der Länge des Wertteils entspricht. Die Bytes 2 und 3 enthalten den Variablennamen in der Reihenfolge 'zweiter Buchstabe/erster Buchstabe'.

Im 1. Abschnitt (einfache Variable) folgt darauf der Wert in der Folge 'LSB........MSB' oder im Falle von Text-Variablen (Strings) 'Länge' und 'Adresse' auf den String im String-Bereich oder in der Programm-Tabelle.

Einträge im Abschnitt 2 haben nach dem 3-Byte Vorspann einen weiteren Kopfeintrag, der Angaben über die Größe der Matrix enthält.

Die Variablen werden während des Programmablaufs in die Variablentabelle eingetragen und zwar immer dann, wenn eine Wertzuweisung erfolgt oder eine Matrix per DIM-Anweisung erstellt wird.

Innerhalb der Abschnitte werden neu auftretende Variable einfach angefügt, es besteht keine alphabetische Sortierfolge. Da Einträge erst dann vorgenommen werden, wenn eine Variable während des Programmablaufs auftritt, kann es geschehen, daß der gesamte 2. Abschnitt im Speicher verschoben werden muß, um Platz für eine einfache Variable im 1. Abschnitt zu schaffen.

Beispiel: Im Programmablauf wurden die Variablen A, B und C(5) bereits definiert und in die Variablen-Tabelle eingetragen.

Folgt nun eine Variable D, so muß die Matrix C(5) im zweiten Abschnitt der Tabelle verschoben werden, so daß D am Ende des ersten Abschnitts eingefügt werden kann.

Matrizen werden im Abschnitt 2 so gespeichert, daß die Indizes von links nach rechts bearbeitet werden

Beispiel: DIM E(2,3) würde wie folgt gespeichert:

E(0,0)
E(1,0)
E(2,0)
.
E(0,3)
E(1,3)
E(2,3)

Die Stellung jedes Elements in der Matrix kann mit folgender Formel ermittelt werden:

(Beispiel für eine drei-dimensionale Matrix)

INDEX = (DRI * GMI + DMI) * GLI + DLI

GMI = Grenzwert mittlerer Index + 1
GLI = Grenzwert linker Index + 1
DRI = definierter rechter Index
DMI = definierter mittlerer Index
DLI = definierter linker Index

Beispiel: Eine Matrix wurde mit DIM A(5,4,4) definiert.

Gesucht wird das Element A(3,1,2).

Das ergibt: GMI = 5, GLI = 6
DRI = 2, DMI = 1, DLI = 3

INDEX = (2 * 5 + 1) * 6 + 3

= 69

A(3,1,2) ist das 69. Element der Matrix.

Die Routine zur Berechnung der Indizes finden Sie im ROM bei Adresse 2795H.

Beispiele für verschiedene Einträge in der Variablen-Tabelle

1. Einfache Variable

Wertlänge	02	
Name	00	
	43	C% = 100
Wert	64	
	00	
Wertlänge	04	
Name	00	
	44	D = - 4
Wert	00	
	00	
	80	
	81	
Wertlänge	03	
Name	00	
	41	
Länge	03	A$ = "XYZ"
Adresse	nn	
	nn	

2. Ein-dimensionale Matrix

DIM A (20)

```
                ---------------------
Wertlänge       !        04         !
Matrixname      !        00         !
                !        41         !
Länge der       !        nn         !     = Distanz zur n. Matrix
Matrix          !        nn         !
Anz. Indizes    !        01         !
Max. Index+1    !        15         !
                !        00         !
A(0)            !        LSB        !
                !        NSB        !
                !        MSB        !
                !        EXP        !
A(1)            !        LSB        !
                !        NSB        !
                !        MSB        !
                !        EXP        !
                .                   .
                .                   .
                .                   .
A(20)           !        LSB        !
                !        NSB        !
                !        MSB        !
                !        EXP        !
                ---------------------
nächste         !        04         !
Matrix          !        00         !
                !                   !
```

3. Drei-dimensionale Matrix

```
                    DIM  A (4,5,9)

                    --------------------
Wertlänge         !      04           !
Matrixname        !      00           !
                  !      41           !
Länge der         !      nn           !    = Distanz zur nächsten Matrix
Matrix            !      nn           !
Anz. Indizes      !      03           !
Max.Wert          !      0A           !
recht.Index+1     !      00           !
Max.Wert          !      06           !
mittl.Index+1     !      00           !
Max.Wert          !      05           !
link.Index +1     !      00           !
A(0,0,0)          !      LSB          !    300 4-Byte Einträge
                  !      NSB          !
                  !      MSB          !
                  !      EXP          !
A(1,0,0)          .                   .
                  .                   .
                  .                   .
A(4,5,9)          !      LSB          !
                  !      NSB          !
                  !      MSB          !
                  !      EXP          !
                  --------------------
nächste           !                   !
Matrix            !                   !
```

10. Die Nutzung des BASIC - Stacks

Vor dem String-Bereich liegt am Ende des für BASIC verfügbaren Speichers der Stack-Bereich des BASIC-Interpreters.

Im Normalfall dient der Kommunikationsbereich dem BASIC-Interpreter als Merkzettel zur Zwischenspeicherung von Arbeitswerten und Adressen. Das geht jedoch nicht immer gut, da einige Routinen sich eventuell selbst intern aufrufen (Rekursion) und dabei die zuvor gespeicherten Zwischenresultate überschrieben würden. Eine indizierte Tabelle würde hier auch Abhilfe schaffen, aber der BASIC-Interpreter bedient sich dazu des Stacks.

Neben der normalen Zwischenspeicherung von Registerinhalten und Rücksprungadressen wird der Stack vom BASIC-Interpreter vornehmlich für drei Sonderfunktionen benutzt:

- FOR/NEXT - Schleifen
- GOSUB - Aufrufe
- Ausdrucksanalyse

Stack-Nutzung in einer FOR/NEXT - Schleife

Bei Auftreten einer FOR-Anweisung werden alle benötigten Variablen-Adressen in einem FOR-Block auf den Stack gepackt. Bei Auftreten einer NEXT-Anweisung wird der FOR-Block mit der entsprechenden Laufvariablen auf dem Stack gesucht. Diese Suchroutine befindet sich bei 1936H.

Der Stack wird dabei von hinten nach vorne durchsucht. Wird kein passender FOR-Block gefunden, so wird die Fehlermeldung

NEXT WITHOUT FOR

erzeugt.

Format eines FOR-Blocks:

```
-------------------
!       81        !   FOR - TOKEN
-------------------
!       LSB       !   Adresse der
!       MSB       !   Laufvariablen
-------------------
!       LSB       !
!        .        !   Erhöhungswert
!        .        !
!       MSB       !
-------------------
!       LSB       !
!        .        !   Endwert
!        .        !
!       MSB       !
-------------------
!       LSB       !   Zeilennummer der
!       MSB       !   FOR-Anweisung (binär)
-------------------
!       LSB       !   Adresse der ersten
!       MSB       !   Schleifenanweisung
-------------------
```

Stack-Nutzung bei einer GOSUB - Anweisung

Bei Auftreten einer GOSUB-Anweisung wird ein 7-Byte Block auf den Stack geschrieben, der von einem anschließenden RETURN ermittelt und ausgewertet wird.

Format eines GOSUB-Blocks:

```
-------------------
!       91        !   GOSUB-TOKEN
-------------------
!       LSB       !   Zeilennummer der
!       MSB       !   GOSUB-Anweisung
-------------------
!       LSB       !   Adresse der GOSUB-Anweisung
!       MSB       !   in der Programm-Tabelle
-------------------
```

11. Die Ausdrucksanalyse

Die Ausdrucksanalyse zerlegt Ausdrücke in ihre einzelnen Elemente und verknüpft diese entsprechend der Rangfolge der Operatoren innerhalb des Ausdrucks.

Jeder Ausdruck wird dabei durchsucht und die höchstwertigste Operation zuerst ausgeführt. Das daraus resultierende Zwischenergebnis wird zwischengespeichert und die nächsthöhere Operation im Ausdruck ermittelt und ausgeführt. Das wird solange fortgesetzt, bis der Ausdruck vollständig aufgelöst wurde.

Ein Ausdruck wird von links nach rechts durchsucht. Die Suche wird unterbrochen, wenn ein Operator oder das Ende des Ausdrucks gefunden wurde. Die Variable links des gefundenen Operators (als aktuelle Variable bezeichnet), zusammen mit dem Operator (ein arithmetisches Verknüpfungssymbol +, -, *, /, [ ) werden als Satz bezeichnet und entweder

- wenn der Rang des Operators größer als der vorausgegangene war, als Satz auf den Stack geschrieben oder

- wenn der Rang des Operators gleich oder niedriger als der vorausgegangene war, die Variable mit dem vorherigen Satz vom Stack verknüpft.
  Der vorherige Satz wird dabei vom Stack entfernt und das Ergebnis der Verknüpfung als neue 'aktuelle Variable' betrachtet.

Das wird solange wiederholt, bis ein so geschaffener neuer Satz auf den Stack 'gepushed' wurde oder keine weiteren Werte zur Verknüpfung mehr auf dem Stack stehen. In einem solchen Falle wurde der Ausdruck vollständig aufgelöst.

Die Variablen/Operator - Sätze werden in folgendem Format auf den Stack geschrieben:

```
                          -----------------
Rang-Wert des voraus-     !       ! XXXXXX !
gegangenen Operators      !       ! XXXXXX !
(beim 1.Eintr. = 0)       !       ! XXXXXX !
                          -----------------
Fortsetzungsadresse       !                !
nach einer Verknüp-       !                !
fung (meist 2346H)        !                !
                          -----------------
Wert der Variablen        !                !
                          !                !
                          !                !
                          -----------------
Typ-Code der              !       !        !   TOKEN des Operators nach der
Variablen                 !       !        !   Variablen (0=+, 1=-, 2=*, 3=/
                          !       !        !   4=[, 5=AND, 6=OR)
                          -----------------
Adresse der Ver-          !                !
knüpfungsroutine          !                !
(für + - * / = 2406)      !                !
                          -----------------
Rang-Wert des             !       ! XXXXXX !
Operators                 !       ! XXXXXX !
                          !       ! XXXXXX !
                          -----------------
```

Die Prüfung, ob eine Verknüpfung stattfinden soll oder nicht, ist relativ einfach. Der Rang-Wert des Operators im letzten Satz ist der letzte Eintrag auf dem Stack. Es wird geprüft, ob der neue Operator im Rang-Wert gleich oder kleiner ist. Wenn nicht, wird der neue Operator und die aktuelle Variable als neuer Satz auf den Stack geschrieben. Wenn ja, wird eine Verknüpfung durchgeführt.

Im Falle einer Verknüpfung wird der letzte Satz vollständig vom Stack geholt und die dort angegebene Verknüpfungsroutine (normalerweise bei 2406H) angesprungen. Dort wird die vorherige Variable vom Stack mit der aktuellen Variablen entsprechend dem vorherigen Operator verknüpft. Das Ergebnis wird die neue aktuelle Variable, die mit dem aktuellen Operator einen neuen Satz bildet.

Im Anschluß an die Verknüpfung wird dorthin zurückgesprungen, wo erneut geprüft wird, ob der jetzt gebildete neue Satz im Rang-Wert kleiner oder gleich dem eines weiteren Satzes auf dem Stack ist.

Ist auf dem Stack kein Satz mehr vorhanden oder hat der vorhandene einen niedrigeren Rang-Wert als der aktuelle Satz, so wird der neu gebildete Satz auf den Stack geschrieben. Ansonsten findet wieder eine Verknüpfung statt.

Das Ende einer Anweisung oder das Auftreten eines nicht-arithmetischen TOKEN löst immer eine Verknüpfung aus.

Das folgende Beispiel soll eine solche Ausdrucksanalyse einmal in Einzelschritten darstellen:

```
     Ausdruck:        A = B + C * D / E [ 5
```

Die Suche beginnt mit dem ersten Zeichen rechts des Gleichheitszeichens und endet zunächst am '+' - Zeichen.

'B' und '+' werden als erster Satz auf den Stack geschrieben, da dort bisher noch kein Satz zum Vergleich vorhanden war bzw. die dort bei der Initialisierung gespeicherte 0 einen niedrigeren Rang-Wert vorspiegelt.

Die Suche wird fortgesetzt und erneut beim '*' unterbrochen. Der Variablen/Operator - Satz 'C *' wird als zweiter Satz auf den Stack geschrieben, da der Rang-Wert des Operators '*' größer als der des vorherigen, auf dem Stack befindlichen '+' ist.

Der Stack sieht jetzt wie folgt aus:

```
--------------------
!  00   ! XXXXXX !
!      2346       !--------
!   Wert von B    !
!  04   !   00    !  Satz 1
!      2406       !
!  79   ! XXXXXX  !--------
!      2346       !--------
!   Wert von C    !
!  04   !   02    !  Satz 2
!      2406       !
!  7C   ! XXXXXX  !--------
!                 !
```

Eine erneute Unterbrechung findet bei dem Zeichen '/' statt. Nun ist eine Verknüpfung durchzuführen, da die Rangwerte von '*' (auf dem Stack) und '/' (der neue Operator) gleich sind.

Satz 2 wird vom Stack gelesen und es wird zur Verknüpfungsroutine bei 2406H verzweigt. Dort wird die Verknüpfung zwischen 'C *' und der aktuellen Variablen 'D' durchgeführt, d.h. 'C' wird mit 'D' multipliziert. Das ergibt eine neue aktuelle Variable, eben das Produkt von 'C * D'.

Nach der Multiplikation wird das Programm bei 2346H fortgesetzt und der neue Variablen/Operator - Satz 'C*D /' im Rang-Wert mit dem auf dem Stack befindlichen Satz 1 verglichen. Da der Rang-Wert von '/' größer ist als der des ersten Satzes ('+'), wird 'C*D /' als 2. Satz auf den Stack gebracht.

Nun hat der Stack folgenden Inhalt:

```
 ________________
!  00   ! XXXXXX !
!      2346      !-------
!   Wert von B   !
!  04   !   00   !  Satz 1
!      2406      !
!  79   ! XXXXXX !-------
!      2346      !-------
!  Produkt C*D   !
!  04   !   03   !  Satz 2
!      2406      !
!  7C   ! XXXXXX !-------
!                !
```

Die Analyse des Ausdrucks wird hinter dem '/' fortgesetzt und hält erneut beim Erreichen des Zeichens '[' an. Der aktuelle Variablen/Operator-Satz 'E[' wird als Satz 3 auf den Stack gebracht, da der Rang-Wert von '[' größer ist als der des vorausgegangenen '/'.

Das ergibt folgendes Bild auf dem Stack:

```
! Satz 1 u.Satz 2 !
!    wie zuvor    !
!      2346       !-------
!   Wert von E    !
!  04   !   04    !  Satz 3
!      2406       !
!  7F   ! XXXXXX  !-------
```

Bei Fortsetzung des Suchvorgangs wird hinter der Ziffer '5' das Ende des Ausdrucks festgestellt. Wie zuvor gesagt, löst das Erreichen des Ausdruckendes eine Verknüpfung aus.

Satz 3 wird vom Stack geholt und 'E [5' berechnet. Das Ergebnis ist die neue aktuelle Variable.

Da der aktuelle Operator immer noch das Erreichen des Ausdruckendes ist, werden nacheinander auch noch Satz 2 und Satz 1 vom Stack geholt und die Operationen

C * D / E [5

und zuletzt

B + C * D / E [5

durchgeführt.

Wird anschließend das Programm bei 2346H fortgesetzt, ist kein weiteres Element des Ausdrucks mehr auf dem Stack und es wird zur aufrufenden Routine zurückgesprungen.

Der Ausdruck wurde ausgewertet, das Ergebnis befindet sich im Arbeitsbereich 1 des Kommunikationsbereichs (X-Register) und wird von dort der Variablen 'A' zugewiesen.



12. Funktionsableitungen

Der BASIC-Interpreter unterstützt 16 arithmetische, davon die folgenden 7 mathematischen Funktionen.

Sinus	(sin)	(a)
Exponentialfunktion	(e)	(b)
Arkus Tangens	(arctan)	(c)
natürl.Logarithmus	(ln)	(d)
Cosinus	(cos)	(e)
Wurzelfunktion	(x)	(f)
Tangens	(tan)	(g)

Die Funktionen e-g können durch die Funktionen a-d ausgedrückt werden.

(1) $$\cos \varphi = \sin \left(\varphi + \frac{\pi}{2} \right)$$; φ im Bogenmaß

(2) $$\tan \varphi = \frac{\sin \varphi}{\cos \varphi}$$; φ im Bogenmaß

(3) $$\sqrt{x} = e^{1/2 \ln x}$$; e = Eulersche Zahl (2.71 .....)

Wird (3) verallgemeinert, ergibt sich:

(4) $$x^y = e^{y \ln x}$$

Zur internen Berechnung verwendet der BASIC-Interpreter arithmetische Näherungen der Funktionen a-d. Alle anderen Funktionen werden mit Hilfe der Näherungen und den Funktionszusammenhängen (1)-(4) errechnet.

Sinus

Nach den Gesetzen der Potenzreihenentwicklung kann der Sinus in einer Reihe ausgedrückt werden.

(5) $$\sin\varphi_0 = \sum_{n=0}^{\infty} (-1)^n \frac{\varphi^{2n+1}}{(2n+1)!}$$

$|\varphi| \leqslant \pi / 2$
Entwicklungspunkt $\varphi_0 = 0$
im Bogenmaß
$|f(x)|$ Betrag $f(x)$

$$= \frac{\varphi^1}{1} - \frac{\varphi^3}{1*2*3} + \frac{\varphi^5}{1*2*3*4*5} - \ldots\ldots$$

Der BASIC-Interpreter nähert den Sinus bis zum 5. Glied. Der Winkel wird im Vielfachen des Kreisbogens (t) angegeben.

(5.1) $\varphi = 2\pi t$; φ im Bogenmaß

(5.2) $$\sin(2\pi t) = 2\pi t - \frac{(2\pi)^3}{3!} t^3 + \frac{(2\pi)^5}{5!} t^5 - \frac{(2\pi)^7}{7!} t^7 + \frac{(2\pi)^9}{9!} t^9$$

Wird x in Grad angegeben, ist ebenfalls t zu errechnen.

(5.3) $$t = \frac{\varphi}{2\pi} = \frac{x}{360^0}$$;x in Grad, φ im Bogenmaß, $|t| \leqslant 1/4$

Zur Vorzeichenbestimmung bei Winkeln zwischen 0 und 360 Grad ($|\varphi| < 2\pi$) errechnet sich t:

(5.4) $$t = \frac{x}{360^0}$$; $0^0 < x \leqslant 90^0$

(5.5) $$t = \frac{180^0}{360^0} - \frac{x}{360^0}$$; $90^0 < x \leqslant 270^0$

(5.6) $$t = \frac{x}{360^0} - \frac{360^0}{360^0}$$; $270^0 < x \leqslant 360^0$

Bei größeren Winkeln ist die Anzahl der ganzen Vielfachen eines Kreisbogens abzuziehen und anschließend ist wie oben zu verfahren.

Die Elemente der einzelnen Reihenglieder sind auf vier Dezimalstellen genau.

Der maximale Fehler der Näherung ist < 0.0000035 (für $|t| < 1/4$). So kann das Gesamtergebnis auf 5 Stellen genau angegeben werden.

Exponentialfunktion

Der BASIC-Interpreter berechnet die Exponentialfunktion e^x für alle Werte

$$-88 < x < 88.$$

Die Funktionsnäherung hat zwei Elemente. Das eine Element ist ein ganzzahliger Exponent zur Basis 2, das zweite eine Reihenentwicklung in 8 Gliedern.

(6) $$e^x = 2^{x \log_2 e}$$

(6.1) $$e^x = e^{-t} \left(2^{[\![x \log_2 e]\!]+1} \right)$$; $[\![f(x)]\!]$ = Treppenfunktion größte ganze Zahl in f(x), bekannt als Integerwert.

$$0 < t \leqslant \ln 2$$

(6.2) $$e^x = T e^{-t}$$; $T = 2^{[\![x \log_2 e]\!]+1}$

Hierbei beschreibt e^{-t} den Unterschied zwischen e^x und dem nächstgrößeren ganzzahligen Vielfachen von $\log_2 e$ als Exponent.

(6.3) $$t = -x + [\![x \log_2 e]\!] \ln 2 + \ln 2$$

(6.4) $$x = \ln e^x = \ln (e^{-t}(2^{[\![x \log_2 e]\!]+1})$$

$$= -t + \ln 2 \, ([\![x \log_2 e]\!]+1) \quad ; \; 0 < t \leqslant \ln 2$$

Die ganzzahlige Potenz zur Basis 2 kann direkt bestimmt werden (binäre Rechensysteme).

Die Potenzreihe für das zweite Element lautet allgemein:

$$(6.5) \qquad e^{-t} = 1 + \sum_{n=0}^{\infty} \frac{(-t)^{n+1}}{(n+1)!}$$

$$= 1 - x + \frac{t^2}{1*2} - \ldots\ldots\ldots$$

Die gefundenen Ergebnisse aus (6.5) und (6.2) ergeben e^x auf 5 Dezimalstellen genau.

Arkustangens

Die Näherung des Inversen Tangens basiert auf der Reihenentwicklung bis zum 9. Glied.

$$(7.1) \qquad \arctan x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)}$$

$$= x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} \ldots\ldots$$

Bei negativen Eingangswerten wird mit dem Betrag von x gerechnet und anschliessend das Ergebnis invertiert. Für Werte $x > 1$ ist die Näherung für $1/x$ zu errechnen.
Das Ergebnis ergibt sich aus:

$$(7.2) \qquad \arctan x = \frac{\pi}{2} - \arctan \frac{1}{x} \qquad ; |x| > 1$$

Für die Werte $!x! < 1$ ergibt sich das Ergebnis direkt aus der Reihenentwicklung. Die Koefizienten 5-7 wurden so korrigiert, daß der maximale Fehler 0.026 (im Bogenmaß) beträgt.

$$(7.3) \qquad \arctan x = x - 0.33331\, x^3 + 0.199936\, x^5 - 0.142089\, x^7 + 0.106563\, x^9 - 0.0752896\, x^{11} + 0.0429096\, x^{13} - 0.01616157\, x^{15} + 0.00286623\, x^{17}$$

Natürlicher Logarithmus

Die Näherung des natürlichen Logarithmus wird aus drei Gliedern der zugehörigen Reihe errechnet.

$$\text{(8.1)} \qquad \ln \left(\frac{1+a}{1-a} \right) = \sum_{n=0}^{\infty} \frac{1}{2n+1} a^{2n+1} \qquad ; \text{ für } |a| < 1$$

$$\text{(8.2)} \qquad x = \frac{1+a}{1-a} \qquad ; \text{ für } |a| < 1$$

Daraus ergibt sich:

$$\text{(8.3)} \qquad \ln x = 2 \left[\frac{x-1}{x+1} + \frac{1}{3} \left(\frac{x-1}{x+1} \right)^3 + \frac{1}{5} \left(\frac{x-1}{x+1} \right)^5 \ldots . \right]$$

Diese Reihe konvergiert für Werte x < 1. Daher muß durch eine Skalierung x in diese Bereiche übertragen werden.

$$\text{(8.4)} \qquad x = y\, 2^n \qquad ; 1/2 \leq y < 1 \; , \; n = 0,1,2 \ldots .$$

Der Skalierungsfaktor ist die nächsthöhere Zweierpotenz mit ganzzahliggem Exponenten.

$$\text{(8.5)} \qquad \ln x = \ln (\, y\, 2^n \,) = \ln y + n \ln 2$$

$$\text{(8.6)} \qquad \ln x = \left(\frac{\ln y}{\ln 2} + n \right) \ln 2$$

$$= \ln 2 \left(\frac{\ln \left(\frac{y}{\ln 2} \right)}{\ln 2} + \frac{\ln (\ln 2)}{\ln 2} + n \right)$$

Die Konstanten dieser Formel werden intern fest angegeben:

$$\ln 2 = 0.0707092$$

$$\frac{\ln(\ln 2)}{\ln 2} = -0.5$$

$$A = \frac{\ln\left(\frac{y}{\ln 2}\right)}{\ln 2} \qquad (8.7)$$

Der skalierte Wert 'A' kann mit der Näherung (8.8) errechnet werden.

$$A = \frac{2}{\ln 2}\left(\frac{\frac{y}{\ln 2} - 1}{\frac{y}{\ln 2} - 1} + \frac{1}{3}(\ldots.)^3 + \frac{1}{5}(\ldots.)^5\right) \qquad (8.8)$$

$$\ln x = 0.707092\,(A - 0.5 + n) \qquad (8.9)$$

Die Genauigkeit wird mit vier Stellen angegeben. Für Werte von x in der Nähe 0 bzw. für sehr große Werte kann diese Genauigkeit nicht erreicht werden.

13. Unterroutinen des BASIC-Interpreters

Der BASIC-Interpreter der LASER-Computer 110, 210, 310 und des VZ200 besteht aus einer großen Anzahl in sich abgeschlossener Routinen, die zur Ausführung bestimmter Funktionen von der Ausführungssteuerung aufgerufen werden. Viele dieser Routinen lassen sich auch von Maschinenprogrammen nutzen und vereinfachen so die Programmerstellung.

In diesem Kapitel werden eine Reihe dieser Routinen beschrieben und anhand von Aufrufbeispielen ihre Einbindung in Maschinenprogramme gezeigt.

Wichtig bei der Nutzung jeder einzelnen Routine ist die genaue Kenntnis der Eingangs- und Ausgangs-Voraussetzungen. Wo sind die Eingangsparameter bereitzustellen, in welchem Datenformat müssen sie bereitgestellt werden und wo und wie wird das Ergebnis bereitgestellt? Das sind nur einige der aufkommenden Fragen. Es ist auch wichtig zu wissen, welche Register von der aufzurufenden Routine verändert werden, damit man sie rechtzeitig vorher sichern kann.

In diesem Zusammenhang sei auch nochmals daran erinnert, daß bei Anschluß eines Diskettensystems das Registerpaar IY nicht verändert werden darf.

Ein-/Ausgabe-Routinen

In diesem Abschnitt werden Routinen beschrieben, die sich mit der Kommunikation des Interpreters mit seiner Umwelt befassen.

Wie können Zeichen von der Tastatur eingelesen, auf dem Bildschirm dargestellt oder auf einen Drucker ausgegeben werden? Wie kann man aus Maschinenprogrammen heraus das Kassetten-Interface nutzen oder Töne auf dem kleinen eingebauten Lautsprecher ausgeben?

Einlesen von der Tastatur

Selbstverständlich bleibt es Ihnen unbenommen, die Tastaturmatrix selbst auszuwerten. Es stehen aber auch ROM-Routinen zur Verfügung, die Ihnen diese Arbeit abnehmen.

CALL 2BH — Tastatur auswerten

Von dieser Routine wird die Tastaturmatrix einmal ausgewertet und das Ergebnis übergeben. Bei gedrückter Taste wird der ASCII-Code ermittelt und ins A-Register eingetragen.

Es wird sofort in die aufrufende Routine zurückgesprungen, unabhängig davon, ob eine Taste betätigt wurde oder nicht. Wollen Sie mehrere Zeichen direkt hintereinander einlesen, so müssen Sie für eine Entprellung selbst Sorge tragen.

Verändert wird das Registerpaar DE und das A-Register, in dem das Ergebnis übermittelt wird.

Beispiel:

```
                .
                .
                LD      BC,600H     ;Möglichkeit der
                CALL    60H         ;Entprellung
                PUSH    DE          ;DE sichern
                CALL    2BH         ;Tastatur auslesen
                OR      A           ;wurde Taste betätigt?
                JP      Z,NEIN      ;==> zum Nein-Zweig
        JA      POP     DE          ;Ja-Zweig
                .
                .
```

Das A-Register enthält beim Rücksprung den ASCII-Code der gedrückten Taste. Wurde keine Taste betätigt, so ist das A-Register leer (00H). Im obigen Beispiel wird die Leseroutine aufgerufen und, abhängig davon, ob eine Taste betätigt wurde oder nicht, entweder zur 'NEIN'-Routine verzweigt oder das Programm bei der 'JA'-Routine fortgesetzt.

'CALL 2BH' bietet sich immer dann an, wenn man während des Programmablaufs im Vorbeigehen schauen will, ob jemand an der Tür geklingelt hat, d.h. eine Taste betätigt wurde.

CALL 49H

Warten auf eine Tastatureingabe

Hier wird Ihnen die Prüfung abgenommen, ob eine Taste betätigt wurde oder nicht. Sie erhalten die Kontrolle erst dann zurück, wenn jemand tatsächlich eine Taste gedrückt hat. Ansonsten entspricht die Routine dem Aufruf 'CALL 2BH', die intern auch benutzt wird.

Beispiel: Auf eine ausgegebene Frage soll der Nutzer mit 'J' = Ja oder 'N' = Nein antworten.
Das Programm wartet auf die Betätigung einer dieser beiden Tasten und verzweigt entsprechend zur Ja- oder Nein-Routine.

```
               .
               .
LES      PUSH  DE         ;DE sichern
         CALL  49H        ;warten auf Tastendruck
         POP   DE         ;DE wiederherstellen
         CP    'J'        ;'J'-Taste betätigt?
         JR    Z,JA       ;==> zum JA-Zweig
         CP    'N'        ;war es die 'N'-Taste?
         JR    NZ,LES     ;auch nicht, weiter warten
NEIN     .                ;Nein-Zweig
         .
         .
JA       .                ;JA-Zweig
         .
         .
```

Die bei 'CALL 2BH' oder 'CALL 49H' eingelesenen Zeichen werden nicht auf dem Bildschirm dargestellt. Das müssen Sie mit einer der nachfolgenden Ausgaberoutinen ggf. selbst bewerkstelligen.

CALL 3E3H

Einlesen einer Zeile

Komfortabler als von den beiden vorherigen Aufrufen werden Sie von der Routine bei 3E3H bedient.

Dort wird eine vollständige Zeile von der Tastatur eingelesen und auf dem Bildschirm dargestellt. Die eingelesene Zeile wird anschließend im Ein-/Ausgabepuffer des Kommunikationsbereichs zur weiteren Verarbeitung bereitgestellt.

Die Darstellung auf dem Bildschirm beginnt an der augenblicklichen Cursor-Position und es wird automatisch das blinkende Cursorzeichen ausgegeben.

Eine Eingabezeile kann maximal 64 Zeichen umfassen und ist mit der <RETURN>-Taste oder den <CTRL-BREAK>-Tasten abzuschließen.

Da diese Routine u.a. zum Einlesen von BASIC-Programmen benutzt wird, sind auch nur die darin gültigen Zeichen zulässig (Buchstaben, Ziffern, Sonderzeichen). Wollen Sie auch Blockgrafikzeichen oder invertierte Zeichen einlesen, so müssen Sie diese bei der Eingabe in Anführungszeichen einschließen. Bei Eingabe falscher Zeichen erscheint der Text "SYNTAX ERROR" und die Eingabe kann wiederholt werden.

Zur Funktion dieser Routine ist es erforderlich, daß die Interrupts eingeschaltet sind (EI = enable interrupts).

Nach einer Eingabe steht der eingelesene Text im Ein-/Ausgabepuffer ab Adresse 79E8H. Das Registerpaar HL zeigt auf das davorliegende Byte (79E7H). Das Textende ist durch ein Zeichen 00H gekennzeichnet.

Anhand des Carry-Flags können Sie ermitteln, ob der Text mit <RETURN> oder <CTRL-BREAK> abgeschlossen wurde.

Carry = 0	mit <RETURN> abgeschlossen
Carry = 1	mit <CTRL-BREAK> abgeschlossen

Alle Register werden von der Routine verändert.

Beispiel: Eine Textzeile ist von der Tastatur einzulesen und ins Feld 'TEXT' zu übertragen. Register B soll anschließend die Textlänge anzeigen.

```
        .
        .
        EI                ;Interrupts einschalten
LES     CALL  3E3H        ;Zeile einlesen
        JR    C,LES       ;BREAK - noch einmal
        INC   HL          ;HL auf Pufferanfang
        LD    DE,TEXT     ;DE = Textbereich
        LD    B,0         ;Zeichenzähler = 0
TRF     LD    A,(HL)      ;1 Zeichen aus Puffer laden
        OR    A           ;Textende ?
        JR    Z,FINI      ;ja, fertig
```

```
                LD      (DE),A          ;Zeichen in Textbereich
                INC     HL              ;Pufferadresse + 1
                INC     DE              ;Textbereich + 1
                INC     B               ;Zeichenzähler + 1
                JR      TRF             ;nächstes Zeichen
FINI            .
                .
                .
TEXT            DEFS    64              ;Textbereich
```

Bei FINI steht die eingegebene Zeile im Feld 'TEXT'
Register B enthält die Textlänge.

Zeichen auf dem Bildschirm darstellen

Drei Routinen bieten sich an, Text auf dem Bildschirm darzustellen. Bei 33AH können Sie ein einzelnes Zeichen ausgeben, bei 28A7H und 2B75H jeweils eine ganze Zeichenkette.

Der Bildschirmspeicher für die Textdarstellung befindet sich bei 7000H-71FFH. Sie können auch direkt etwas in diesen Bereich schreiben, sollten aber die Bildschirm-Synchronisation dabei beachten; doch auch diese Möglichkeit wird in einem Beispiel dargestellt.

CALL 33AH — Ein Zeichen darstellen

Das im A-Register bereitgestellte Zeichen wird an der Cursor-Position auf dem Bildschirm dargestellt. Der Cursor wird um ein Zeichen vorgesetzt.

Im A-Register sind dabei die normalen ASCII-Codes zu verwenden, nicht die LASER-internen Bildschirm-Codes.

Neben den alphanumerischen Zeichen werden auch Steuerzeichen der ASCII-Codetabelle erkannt und die entsprechenden Bildschirmfunktionen ausgeführt.

08H o. 18H	Cursor eine Stelle nach links
09H o. 19H	Cursor eine Stelle nach rechts
0AH	Cursor eine Zeile nach unten
0DH	Cursor an den Anfang der nächsten Zeile
15H	Ein Leerzeichen einfügen (INSERT)
1BH	Cursor eine Zeile nach oben
1CH	Cursor in die linke obere Ecke
1DH	Cursor an den Zeilenanfang
1FH	Bild löschen
7FH	Zeichen an der Cursorposition löschen (RUBOUT)

Registerinhalte werden nicht verändert, sie werden von der Routine beim Ansprung gesichert und beim Rücksprung wieder restauriert.

Beispiel: An der Cursorposition soll der Buchstabe 'A' dargestellt werden.

```
        .
        .
        LD      A,'A'       ;Zeichen laden
        CALL    33AH        ;und ausgeben
        .
        .
```

CALL 28A7H Eine Zeile ausgeben
oder
CALL 2B75H

Haben Sie mehr als ein Zeichen auszugeben, so könnten Sie den Aufruf bei 33AH in einer Schleife mehrfach durchlaufen. Die Routinen bei 28A7H und 2B75H nehmen Ihnen diese Arbeit ab und geben einen vollständigen Text mit einem einzigen CALL-Aufruf auf den Bildschirm aus.

Die Textanfangsadresse muß im HL-Registerpaar bereitgestellt werden, das Textende ist mit 00H zu kennzeichnen, bei 28A7H wird auch 0DH als Textende erkannt.

Die Routine bei 28A7H benutzt zur Textaufbereitung den String-Zwischenspeicher im Kommunikationsbereich und den BASIC-Stringbereich am Speicherende. Sie sollten daher diese Routine nur dann benutzen, wenn Kommunikations- und Stringbereich von Ihrem Maschinenprogramm sauber verwaltet werden.

Die Routine bei 2B75H führt eine direkte Ausgabe durch und unterliegt nicht den o.a. Restriktionen. Mit dieser Routine wird Ihnen noch ein weiterer Komfort geboten. Durch einfache Umschaltung eines Bytes können Sie die Ausgabe vom Bildschirm auf einen angeschlossenen Drucker oder sogar auf den Kassettenrekorder umleiten. Dies ist das Byte 789CH im Kommunikationsbereich.

00H = Bildschirm
01H = Drucker
80H = Kassette

Vergessen Sie aber anschließend nicht, dieses Byte wieder auf die Bildschirmausgabe zu setzen. Sonst wandern evtl. spätere Systemausgaben auch auf das andere Gerät.

Beispiel: Der Bildschirm soll gelöscht und anschließend der Text 'MEIN LASER IST PHANTASTISCH' ausgegeben werden.

```
        .
        .
        LD      HL,TEXT         ;Textadresse laden
        CALL    2B75H           ;Text ausgeben
        .
        .

TEXT    DEFB    1FH             ;Bildschirm löschen
        DEFM    'MEIN LASER IST PHANTASTISCH'
        DEFB    0               ;Textendekennung
```

CALL 1C9H — Löschen des Bildschirms

Den Bildschirm sauber bekommen Sie einfach mit einem CALL 1C9H. Intern geschieht dabei nicht mehr, als die Steuerzeichen 1DH (Cursor an den Bildanfang) und 1FH (Bild löschen) mit zwei aufeinanderfolgenden CALL 33AH auszugeben.

Beispiel:

```
        .
        .
        CALL    1C9H            ;Bildschirm löschen
        .
        .
```

Direktausgabe in den Bildschirmspeicher

Wollen Sie direkt in den Bildschirmspeicher (7000H-71FFH) schreiben, so dürfen Sie nicht den normalen ASCII-Zeichensatz verwenden. Das auszugebende Zeichen muß in der LASER-internen Verschlüsselung bereitgestellt werden, dafür können aber auch Blockgrafikzeichen aller Farben direkt ausgegeben werden (siehe Kapitel 5). Das Verfahren eignet sich auch für die Ausgabe in der hochauflösenden Grafik (7000H-77FFH). Beachten Sie dabei aber, daß jedes Byte die Information für vier Pixels enthält.

Eine solche Ausgabe sollte jedoch mit dem Bildgenerator synchronisiert werden, da ansonsten die Bildqualität darunter leidet (bei häufigen Ausgaben wird das Bild durch Querstriche gestört).

Eine Möglichkeit der Synchronisation bietet der RAM-Erweiterungsausgang der Interrupt-Service-Routine bei 787DH. Dort können Sie einen Sprung 'C3 xx xx' zu einer eigenen Routine einsetzen. Nach Ausgabe Ihres Textes in dieser Routine kehren Sie mit einem einfachen RETURN (C9H) in die normale Interrupt-Service-Routine zurück.

Ein andere Möglichkeit, bei ausgeschaltetem Interrupt (DI), bietet die Abfrage des Bits 7 im Ein-/Ausgabebereich 6800H-6FFFH (siehe Kapitel 4). Dieses Bit ist direkt mit dem Vertikal-Sync-Signal des Bildgenerators verbunden.

Beispiel: Am Anfang der 3. Zeile des Bildschirms soll der Buchstabe 'A' ausgegeben werden.

```
               .
               DI                  ;Interrupts ausschalten
               LD    HL,7040H      ;3. Zeile adressieren
               LD    B,'A'         ;auszugebendes Zeichen laden
        WARTEN LD    A,(6800H)     ;Vertikal-Sync prüfen
               OR    A
               JP    P,WARTEN      ;wenn Bit 7 = 0, warten
               LD    (HL),B        ;Zeichen ausgeben
               .
               .
```

Zeichen auf den Drucker ausgeben

Aus einem Maschinenprogramm heraus können ROM-Routinen benutzt werden, Zeichen auf einen angeschlossenen Drucker auszugeben. Eine dieser Möglichkeiten wurde bereits bei der Bildschirmausgabe mit dem CALL 2B75H und der Umschaltung des Flags bei 789CH besprochen. Damit können ganze Textzeilen an den Drucker übertragen werden. Eine weitere Möglichkeit wird nachstehend beschrieben.

Bei einem Drucker der Marke "Seikosha GP100" (siehe Kapitel 8) können auch Zeichen der Blockgrafik und invertierte Textzeichen gedruckt werden.

Die Routine zur Ausgabe einzelner Zeichen wartet automatisch auf das "Fertigwerden" des Druckers. Sie haben jedoch auch selbst die Möglichkeit, den Druckerstatus zu erfragen.

CALL 3BH — Ein Zeichen drucken

Das im A-Register übergebene Zeichen (im ASCII-Code) wird auf einen angeschlossenen Drucker ausgegeben.

Im Gerätesteuerblock (Device Control Block) bei 7825H-782CH wird automatisch ein Zeilenzähler mitgeführt, der bei 66 Zeilen jeweils auf 0 zurückgesetzt wird. Diese 66 Zeilen pro Seite sind eine Systemvorgabe bei der Initialisierung des Kommunikationsbereichs und stehen bei Adresse 7828H (Anzahl Zeilen/Seite + 1). 66 Zeilen/Seite entspricht 11 Zoll und damit dem amerikanischen Blattformat. Das deutsche DIN A4 - Format hat ca. 12 Zoll, d.h. bei deutschem Papiermaß sollten Sie, wenn Sie die Blattvorschubsteuerung nutzen wollen, diesen Wert auf 72 Zeilen/Seite (+1 = 49H) ändern.

Neben ganz normalen ASCII-Textzeichen können auch Steuerzeichen an den Drucker übertragen werden, z.B. Escape-Sequenzen zur Schriftwahl (siehe Drucker-Bedienungsanleitung)

Folgende Steuerzeichen werden bereits vom Drucker-Treiber erkannt und ausgeführt:

00H - Der Druckerstatus wird ermittelt und im Bit 0 des A-Registers zurückgegeben.
Bit 0 = 0 - der Drucker ist druckbereit
Bit 0 = 1 - der Drucker ist nicht bereit
Das Zero-Flag wird entsprechend gesetzt.

0BH	-	Absoluter Blattvorschub. Der Drucker wird auf den Anfang der nächsten Seite vorgesetzt.
0CH	-	Bedingter Blattvorschub. Der Drucker wird nur dann an den Anfang der nächsten Seite vorgesetzt, wenn er nicht bereits am Anfang einer Seite steht (Zeilenzähler bei 7829H = 0).
0DH		Wagenrücklauf (Carriage Return - CR) Es wird ein Wagenrücklauf-Zeichen (0DH) und ein Zeilenvorschub-Zeichen (0AH) ausgegeben. Achtung: Sie sollten Ihren Drucker so einstellen, daß er nicht automatisch nach einem Wagenrücklauf einen Zeilenvorschub ausführt (siehe Drucker-Bedienungsanleitung). Ansonsten wird immer eine Leerzeile eingefügt.
0AH		Zeilenvorschub. Wird vor der Ausführung intern in ein 0DH umgewandelt.

CALL 5C4H — Druckerstatus ermitteln

Durch Aufruf dieser Routine kann der Druckerstatus direkt abgefragt werden (nur die 'BUSY'-Leitung wird überwacht).

Im Bit 0 des A-Registers wird der Status übergeben, das ZERO-Flag wird entsprechend gesetzt.

Bit 0 = 0 (Z-Flag = 1) - Der Drucker ist bereit.
Bit 0 = 1 (Z-Flag = 0) - Der Drucker ist nicht bereit.

Beispiel: Der Text 'TEST EINER DRUCKERAUSGABE' soll auf dem Drucker ausgegeben werden.
Ist dieser nicht druckbereit, soll stattdessen auf dem Bildschirm der Text 'DRUCKER NICHT BEREIT' erscheinen.

```
        .
        .
        CALL    5C4H        ;Druckerstatus prüfen
        JR      NZ,FEHLER   ;nicht bereit
        LD      HL,DRTEXT   ;Text adressieren
LOOP    LD      A,(HL)      ;Textzeichen laden
        OR      A           ;Textende?
        JR      Z,FERTIG    ;ja, fertig
        CALL    3BH         ;Zeichen drucken
        INC     HL          ;nächstes Zeichen adressieren
        JR      LOOP        ;und ausgeben
FERTIG  .
        .
        .
FEHLER  LD      HL,FETEXT   ;Fehlertext adressieren
        CALL    2B75H       ;auf Bildschirm darstellen
        .
        .
DRTEXT  DEFM    'TEST EINER DRUCKERAUSGABE'
        DEFB    0
FETEXT  DEFM    'DRUCKER NICHT BEREIT'
        DEFB    0
```

Die Kassetten - Ein-/Ausgabe

Die Kommunikation mit einem angeschlossenen Kassettenrekorder erfolgt über den Ein-/Ausgabebereich 6800H-6FFFH (siehe Kapitel 4). über Bit 6 wird beim Lesen die Information vom Rekorder übernommen, über Bit 1 und 2 erfolgt die Ausgabe auf den Kassettenrekorder. Im Gegensatz zu Bildschirm und Drucker erfolgt die übertragung seriell, d.h. für ein zu schreibendes oder zu lesendes Zeichen (= 1 Byte) müssen 8 Bits hintereinander geschrieben oder gelesen werden.

Es stehen eine Reihe von Routinen zur Verfügung, die Sie bei eigener Bearbeitung der Kassetten-Schnittstelle nutzen können. Das Standardformat einer Aufzeichnung können Sie dem Kapitel 8, Abschnitt "Der Kassettentreiber" entnehmen.

Da die Bitaufzeichnung sehr zeitkritisch ist, ist es wichtig, vor einer Bearbeitung (Lesen oder Schreiben) auf alle Fälle die Interrupts auszuschalten (DI), ansonsten übertragen Sie keine brauchbaren Informationen.

Schreiben auf die Kassette

CALL 3511H

Ein Byte auf die Kassette schreiben

Mit dieser Routine wird ein Byte bitseriell auf den Kassettenrekorder ausgegeben. Das auszugebende Byte ist im A-Register bereitzustellen.

CALL 3558H

Dateivorspann auf Diskette schreiben

Hiermit wird ein vollständiger Dateikopf auf den Kassettenrekorder ausgegeben. Dieser besteht aus den Synchronisationsbytes (255 x 80H), dem Vorspann (5 x FEH), dem Dateikenner (F0H = BASIC, F1H = Binärdatei, F2H = Datendatei) und dem Dateinamen (max. 15 Zeichen).

Der Dateiname ist in einem eigenen Feld, in Anführungsstriche eingeschlossen, bereitzustellen. HL muß die Anfangsadresse dieses Feldes enthalten. Der Dateikenner ist im C-Register zu übergeben.

Ist beim Rücksprung das Carry-Bit gesetzt, so wurde der Aufzeichnungsvorgang durch die <CTRL-BREAK>-Tasten unterbrochen.

Beispiel: Der Speicherbereich von 8000H bis 8FFFH soll als Binärdatei mit dem Namen "TEST" auf den Kassettenrekorder ausgegeben werden. Eine Prüfsumme soll für eine spätere Ladekontrolle hinter die Daten geschrieben werden.

```
        .
        .
        DI              ;Interrupts ausschalten
        LD    C,0F1H    ;Kenner für eine Binärdatei
        LD    HL,NAME   ;Dateinamen adressieren
        CALL  3558H     ;Dateivorspann ausgeben
        JP    C,BREAK   ;BREAK-Taste betätigt
        LD    BC,400    ;eine kleine Pause dazwischen
        CALL  60H
        CALL  3AE8H     ;BREAK-Taste betätigt?
        JP    C,BREAK   ;ja!
        LD    IX,7823H  ;Prüfsummenbytes adressieren
        LD    HL,8000H  ;Startadresse laden
        LD    A,L       ;LSB Startadresse auf Kassette
        CALL  3511H     ;schreiben
        LD    (IX),A    ;und in Pruefsumme übernehmen
```

```
        XOR   A            ;MSB Prüfsumme = 0
        LD    (IX+1),A
        LD    A,H          ;MSB der Startadresse
        CALL  3511H        ;auf Kassette ausgeben
        CALL  388EH        ;und auf Prüfsumme addieren
        EX    DE,HL        ;Startadresse in DE
        LD    HL,8FFFH     ;Endadresse laden
        INC   HL           ;+ 1 für Aufzeichnung
        LD    A,L          ;LSB Endadresse
        CALL  3511H        ;auf Kassette ausgeben
        CALL  388EH        ;und auf Prüfsumme addieren
        LD    A,H          ;MSB Endadresse
        CALL  3511H        ;auf Kassette ausgeben
        CALL  388EH        ;und auf Prüfsumme addieren
LOOP    CALL  3AE8H        ;BREAK-Taste betätigt?
        JP    C,BREAK      ;ja!
        LD    A,(DE)       ;Byte des Speicherbereichs laden
        INC   DE           ;Speicheradresse + 1
        CALL  3511H        ;Byte auf Kassette ausgeben
        CALL  388EH        ;und auf Prüfsumme addieren
        RST   18H          ;Ende erreicht ?
        JR    NZ,LOOP      ;nein, nächstes Byte
        LD    A,(IX)       ;LSB der Prüfsumme
        CALL  3511H        ;auf Kassette ausgeben
        LD    A,(IX+1)     ;MSB der Prüfsumme
        CALL  3511H        ;auf Kassette ausgeben
        EI                 ;Interrupts wieder einschalten
        .
        .
BREAK   .
        .
        .

NAME    DEFM  "TEST"       ;Dateiname
```

Im obigen Beispiel wurden einige Routinen aufgerufen, die bisher nicht beschrieben sind.

CALL 3AE8H

BREAK-Taste abfragen

In dieser Routine wird geprüft, ob die <CTRL>- und <BREAK>-Tasten gleichzeitig betätigt wurden (BREAK-Funktion). Ist dies der Fall, wird das CARRY-Bit gesetzt.

CALL 388EH

Prüfsumme erstellen

Diese Routine wird von der Kassetten-Aufzeichnungsroutine und der -Leseroutine benutzt, um die Prüfsumme einer Aufzeichnung zu ermitteln. Dazu stehen im Kommunikationsbereich die beiden Bytes 7823H und 7824H zur Verfügung, die mit dem Registerpaar IX zu adressieren und zu initialisieren sind.

RST 18H

Vergleich HL mit DE

Hier wird das Registerpaar HL mit dem Registerpaar DE logisch verglichen. Das CARRY- und das ZERO-Flag werden entsprechend dem Ergebnis des Vergleichs gesetzt.

Bei der Behandlung der RESTART-Prozeduren wird diese Routine ausführlich beschrieben.

Lesen von der Kassette

CALL 3775H

Ein Byte von Kassette lesen

Hiermit wird ein einzelnes Byte von der Kassette gelesen und im A-Register zur Verfügung gestellt. Die Registerpaare BC, DE und HL bleiben unverändert.

Bei Lesefehlern wird das CARRY-Bit gesetzt.

Vor dem Lesen eines Bytes muß die Leseroutine auf eine gültige Aufzeichnung synchronisiert werden.

Beim Lesen mehrerer Bytes ist der 600 Baud Rhythmus einzuhalten.

CALL 35E1H Datei auf der Kassette suchen

Mit dieser Routine wird der Anfang einer Datei auf der Kassette ermittelt und die Leseroutine auf die Aufzeichnung synchronisiert.

Der Name der zu suchenden Datei ist im Kommunikationsbereich ab Adresse 7AB2H zu hinterlegen und mit 00H abzuschließen. Dazu kann auch die Routine bei 358CH benutzt werden, der die Anfangsadresse des Namensfeldes in HL zu übergeben ist.

Wird eine Datei des angegebenen Namens auf der Kassette gefunden, so wird die Dateikennung ins Feld 7AB2H übertragen und kann dort abgeprüft werden.

Während des Suchvorgangs werden in der letzten Bildschirmzeile Meldungen über den Suchstatus ausgegeben:

WAITING - es wurden noch keine Synchronisationsbytes gefunden.

FOUND X: Dateiname

Eine Datei mit dem angegebenen Namen wurde gefunden. 'X:' = Dateikenner
- T = Text-Datei (z.B. BASIC-Programm)
- B = Binär-Datei (z.B. Maschinen-Programm)
- D = Daten-Datei

Entspricht der angegebene Namen nicht der gesuchten Datei, so wird der Suchvorgang automatisch fortgesetzt und es erscheinen ggf. mehrere FOUND-Meldungen hintereinander.

Die o.a. Meldungen können unterdrückt werden, indem im Byte 784CH des Kommunikationsbereichs ein Wert ungleich 0 eingetragen wird.

Wird während des Suchvorgangs die BREAK-Taste betätigt (CTRL-BREAK), so wird nicht zum aufrufenden Programm, sondern zur BASIC-Hauptschleife zurückgesprungen.

Beispiel: Der im vorherigen Beispiel aufgezeichnete Speicherbereich soll wieder eingelesen werden. Die Prüfsumme ist zu ermitteln und am Aufzeichnungsende zu überprüfen. Die Meldungsausgabe soll unterdrückt werden.

```
        .
        .
        DI                   ;Interrupts ausschalten
        LD     A,1           ;Meldungsausgabe unterdrücken
        LD     (784CH),A
        LD     HL,NAME       ;Dateinamen in Kommunikations-
        CALL   358CH         ;bereich übertragen
SUCH    CALL   35E7H         ;Datei auf Kassette suchen
        LD     A,(7AD2H)     ;Name gefunden, Kennung prüfen
        CP     0F1H          ;Ist es eine Binär-Datei?
        JR     NZ,SUCH       ;nein, weitersuchen
        LD     IX,7823H      ;Prüfsummenbytes adressieren
        CALL   3868H         ;Start- und Endadresse lesen
        JP     C,FEHLER      ;Lesefehler!
        OR     A             ;Carry löschen
        SBC    HL,DE         ;Programmlänge ermitteln
        JP     C,FEHLER      ;Startadresse > Endadresse
        PUSH   HL            ;Länge in BC übertragen
        POP    BC
LOOP    CALL   3775H         ;Byte von Kassette lesen
        JP     C,FEHLER      ;Lesefehler
        LD     (DE),A        ;in Speicherbereich übertragen
        CALL   388EH         ;und auf Prüfsumme addieren
        INC    DE            ;Speicheradresse + 1
        DEC    BC            ;Länge - 1
        LD     A,C           ;= 0 ?
        OR     B
        JR     NZ,LOOP       ;nein, weiterlesen
        CALL   3775H         ;LSB der Prüfsumme lesen
        CP     (IX)          ;= der errechneten ?
        JP     NZ,FEHLER     ;nein, Ladefehler
        CALL   3775H         ;MSB der Prüfsumme lesen
        CP     (IX+1)        ;= der errechneten ?
        JP     NZ,FEHLER     ;nein, Ladefehler
        EI                   ;Interrupts einschalten, fertig!
        .
        .
FEHLER  .
        .
        .
NAME    DEFM   '"TEST"'      ;Dateiname
```

Im Beispiel wurden zwei Hilfsroutinen benutzt, die noch kurz erläutert werden sollen.

CALL 358CH

Dateinamen übertragen

Die Routine dient dazu, einen im Programm befindlichen Dateinamen in den Kommunikationsbereich ab Adresse 7A9DH zu übertragen.

Der Dateiname ist vor dem Aufruf mit HL zu adressieren und muß in Anführungsstriche eingeschlossen sein.

CALL 3868H

Start- und Endadresse laden

Wurde die richtige Datei auf der Kassette gefunden, so kann hiermit die Start- und Endadresse des Speicherbereichs von der Kassette gelesen werden.

Beim Rücksprung enthält DE die Startadresse und HL die Endadresse + 1 einer Text- oder Binärdatei.

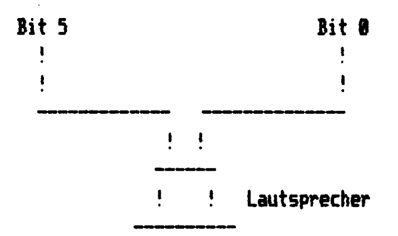
Die Prüfsumme wird von dieser Routine initialisiert.

Ist beim Rücksprung das CARRY-Flag gesetzt, so trat ein Lesefehler auf.

Lautsprecher - Ausgabe

Den kleinen eingebauten Lautsprecher der LASER-Computer und des VZ200 können Sie aus Maschinenprogrammen heraus auch ansprechen.

Dieser Lautsprecher ist, wie eingangs bereits beschrieben, mit den Bits 0 und 5 des Ein-/Ausgabebereichs 6800H-6FFFH fest verdrahtet.



Diese beiden Bits müssen immer komplementär sein. Durch Umschalten der Bits in einer bestimmten Frequenz wird ein Ton erzeugt. Sie brauchen jedoch nicht zu befürchten, dies aus Maschinenprogrammen heraus selbst durchführen zu müssen. Zwei ROM-Routinen erlauben es, entweder einen einzelnen Ton oder eine ganze Melodie mit einem Aufruf auszugeben.

CALL 345CH — Einen einzelnen Ton ausgeben

Beim Aufruf ist im Registerpaar HL die Pulslänge (gemäß Tabelle ab 2CFH) und im Registerpaar BC die Tondauer anzugeben.

Alle Register werden verändert.

Zur Erzeugung eines sauberen Tones sollten die Interrupts ausgeschaltet werden.

Beispiel:

```
        .
        .
        DI                      ;Interrupts ausschalten
        LD      HL,0A0H         ;Pulslänge laden
        LD      BC,6            ;Tondauer laden
        CALL    345CH           ;Ton ausgeben
        EI                      ;Interrupts wieder einschalten
        .
        .
```

Es wird ein hoher kurzer Piepton ausgegeben.

Je höher der Wert in HL ist, umso dunkler wird der Ton. Die Tondauer ist auch von der Tonhöhe abhängig. Um bei zwei unterschiedlichen Tönen die gleiche Tondauer zu erzielen, muß beim höheren Ton ein größerer Wert für BC gewählt werden.

CALL 2BF5H — Eine Melodie spielen

Diese Routine erlaubt das Abspielen einer vollständigen Melodie mit einem Aufruf. Die einzelnen Noten sind dabei wie bei einem BASIC-SOUND-Befehl anzugeben:

Ton,Länge; Ton,Länge; Ton,Länge; ....

Die Melodie ist als ASCII-String im Programm bereitzustellen und mit dem HL-Registerpaar zu adressieren. Ton und Längenangabe sind durch Komma zu trennen, mehrere Töne können, durch Semikolon getrennt, hintereinander angegeben werden.

Beispiel: Spielen einer kleinen Melodie

```
            .
            .
            LD      HL,MELO       ;Melodie-String adressieren
            CALL    3BF5H         ;Melodie spielen
            .
            .
    MELO    DEFM    '16,2;21,2;21,2;23,2;23,2;25,3;26,1;'
            DEFM    '28,2;26,2;25,2;23,2;23,2;21,4'
```

Umwandlungs - Routinen

Datentyp-Umwandlung

Eine Reihe von Unterroutinen dient lediglich dazu, vor einer Verarbeitung die numerischen Daten im richtigen Typ bereitzustellen, d.h. von einem Datentyp in einen anderen umzuwandeln.

Die Umwandlungsroutinen erwarten den umzuwandelnden Wert im Arbeitsregister 1 des Kommunikationsbereichs (X-Register) und den Datentyp dieses Wertes im Typ-Flag bei Adresse 78AFH.

Das Ergebnis wird wieder im Arbeitsbereich 1 zur Verfügung gestellt. Das Typ-Flag 78AFH enthält nach der Umwandlung das Kennzeichen des neuen Datentyps.

CALL 0A7FH — Fließkommazahl in Integer

Der Inhalt des Arbeitsbereichs 1 wird von einer Variablen einfacher oder doppelter Genauigkeit in eine Ganzzahl (Integer) umgewandelt.

Alle Register werden verändert.

Eine Rundung findet nicht statt.

Beipiel: Der Wert 2.88539 soll vom Typ "einfache Genauigkeit" in eine Ganzzahl umgewandelt werden.

```
             .
             .
             LD      DE,AWERT      ;Ausgangswert adressieren
             LD      HL,7921H      ;Arbeitsbereich 1 adressieren
             LD      BC,4          ;Wertlänge = 4
             LDIR                  ;Ausgangswert in Arbeitsbereich 1
             LD      A,4           ;Typ-Flag = einfache Genauigkeit
             LD      (78AFH),A
             CALL    0A7FH         ;Umwandlungsroutine aufrufen
             LD      HL,(7921H)    ;Ergebnis aus Arbeitsbereich 1
             LD      (ZWERT),HL    ;ins Ergebnisfeld übertragen
             .
             .
   AWERT     DEFB    45H           ;Ausgangswert 2.88539
             DEFB    0AAH          ;als Fließkommazahl
             DEFB    38H           ;einfacher Genauigkeit
             DEFB    82H           ;LSB-NSB-MSB-EXP
             .
   ZWERT     DEFW    0             ;Ergebnisfeld
                                   ;enthält nach der Umwandlung
                                   ;den Wert 2
```

CALL 0AB1H   Ganzzahl in Zahl einfacher Genauigkeit

Der Inhalt des Arbeitsbereichs 1 wird von einer Ganzzahl (Integer) in eine Fließkommazahl einfacher Genauigkeit umgewandelt.

Beispiel: Die Zahl 18569 ist vom Format der Ganzzahl in eine Fließkommazahl einfacher Genauigkeit umzuwandeln und ins Feld ZWERT zu übertragen.

```
             .
             .
             LD      A,89H         ;LSB von 18569
             LD      (7921H),A     ;in Arbeitsbereich 1
             LD      A,48H         ;MSB von 18569
             LD      (7922H),A     ;in Arbeitsbereich 1
             LD      A,2           ;Typ-Flag = Integer
             LD      (78AFH),A
```

```
        CALL    0AB1H       ;Wert umwandeln
        LD      HL,ZWERT    ;Ergebnis ins Feld ZWERT
        CALL    9CBH        ;übertragen
        .
        .
ZWERT   DEFS    4           ;Ergebnisfeld
```

ZWERT enthält nach der Umwandlung den Wert
00H-12H-11H-8FH (LSB-NSB-MSB-EXP).
Dies entspricht der Zahl 18569H in der Form als
Fließkommazahl einfacher Genauigkeit.

CALL 0ADBH

Ganzzahl in Zahl doppelter Genauigkeit

Der Inhalt des Arbeitsbereichs 1 wird von einer Ganzzahl (Integer) in eine Fließkommazahl doppelter Genauigkeit umgewandelt.

Beispiel: Die Zahl 657 ist in eine Fließkommazahl doppelter Genauigkeit umzuwandeln.

```
        .
        .
        LD      A,91H       ;LSB von 657
        LD      (7921H),A   ;in Arbeitsbereich 1
        LD      A,2         ;MSB von 657
        LD      (7922H),A   ;in Arbeitsbereich 1
        LD      A,2         ;Typ-Flag = Integer
        LD      (78AFH),A
        CALL    0ADBH       ;in Wert doppelter Ganauigkeit umw
        LD      DE,ZWERT    ;Ergebnis ins Ergebnisfeld
        LD      HL,791DH    ;übertragen
        LD      BC,8        ;(Länge = 8)
        LDIR
        .
        .
ZWERT   DEFS    8           ;Ergebnisfeld
```

ZWERT enthält nach der Umwandlung den Wert 657 als Fließkommazahl doppelter Genauigkeit
(00H-00H-00H-00H-00H-40H-24H-8AH)

ASCII-String in numerische Darstellung

Die nachfolgenden drei Routinen wandeln einen numerischen ASCII-String in einen der drei Datentypen um.

Beim Einsprung muß das Registerpaar HL auf den Anfang des umzuwandelnden Strings zeigen. Die Umwandlung wird bei Erreichen des ersten nichtnumerischen Zeichens beendet.

Alle Register werden verändert.

CALL 1E5AH — ASCII-String in Ganzzahl umwandeln

Der mit HL adressierte ASCII-String wird in eine Ganzzahl (Integer) umgewandelt.

Das Ergebnis wird im Registerpaar DE übergeben.

Beispiel: Der ASCII-String '16544' soll in eine Ganzzahl umgewandelt werden.

```
                .
                .
                LD      HL,AWERT     ;Ausgangswert adressieren
                CALL    1E5AH        ;ASCII-String in Ganzzahl
                LD      (ZWERT),DE   ;Ergebnis ins Ergebnisfeld
                .
                .
        AWERT   DEFM    '16544'      ;Ausgangswert
                DEFB    0            ;Endekennung
                .
        ZWERT   DEFW    0            ;Ergebnisfeld
```

ZWERT enthält nach der Umwandlung den Wert 16544 als 2-Byte binäre Ganzzahl (= 40A0H).

CALL ØE6CH

ASCII-String in Binärwert beliebigen Typs umwandeln

Wandelt den mit HL adressierten numerischen ASCII-String in einen der drei binären Datentypen um.

Ist der Wert des ASCII-Strings kleiner als 32768 und enthält der ASCII-String keinen Dezimalpunkt, keine Exponentenangabe 'E' oder 'D' und keine Typkennzeichnung '#' oder '!', so erfolgt die Umwandlung in eine 2-Byte Ganzzahl (Integer).

Ist der Wert größer als 32767 oder enthält der ASCII-String einen Dezimalpunkt, die Exponentenangabe 'E' oder die Typkennzeichnung '!', so wird er in eine Fließkommazahl einfacher Genauigkeit umgewandelt.

Bei einer Exponentenangabe 'D' oder einer Typkennzeichnung '#' erfolgt die Umwandlung in eine Fließkommazahl doppelter Genauigkeit.

Beispiele:

'12345'	-	Umwandlung in eine Ganzzahl
'40516'	-	Umwandlung in einfache Genauigkeit
'12.3'	-	Umwandlung in einfache Genauigkeit
'12345!	-	Umwandlung in einfache Genauigkeit
'12345#'	-	Umwandlung in doppelte Genauigkeit
'123E10'	-	Umwandlung in einfache Genauigkeit
'123D10'	-	Umwandlung in doppelte Genauigkeit

Das Ergebnis wird im Arbeitsbereich 1 übergeben. Im Typ-Flag wird der Typ des Ergebnisses angezeigt.

Beispiel:

```
        .
        .
        LD      HL,AWERT      ;ASCII-String adressieren
        CALL    0E6CH         ;in Binärwert umwandeln
        .
        .
AWERT   DEFM    '24657'       ;ASCII-String
        DEFB    0             ;Endekennung
```

Der String '24657' wird in eine Ganzzahl umgewandelt und im Arbeitsbereich 1 (7921H-7922H) übergeben. Das Typ-Flag bei 78AFH wird auf 02H (= Ganzzahl) gesetzt.

CALL 0E65H

ASCII-String in doppelte Genauigkeit

Dies ist ein Vorspann vor der o.a. Routine bei 0E6CH. Damit wird eine Umwandlung in eine Fließkommazahl doppelter Genauigkeit erzwungen, unabhängig von der Größe und Konstellation des ASCII-Strings.

Beispiel:

```
        .
        .
        LD      HL,AWERT     ;ASCII-String adressieren
        CALL    0E65H        ;in dopp. Genauigkeit umwandeln
        .
        .
AWERT   DEFM    '24567'      ;ASCII-String
        DEFB    0            ;Endekennung
```

Der String '24567' wird in eine Fließkommazahl doppelter Genauigkeit umgewandelt.

Binär-Wert in ASCII-String umwandeln

Die drei nachfolgenden Routinen wandeln einen numerischen Wert aus dem Binärformat in einen ASCII-String um.

CALL 0FAFH

Inhalt von HL in ASCII umwandeln

Ein im HL-Registerpaar befindlicher Binärwert wird in einen ASCII-String umgewandelt und an der Cursorposition auf dem Bildschirm dargestellt.

Diese Routine wird vom BASIC-Interpreter dazu benutzt, die Zeilennummer einer Programmzeile auf dem Bildschirm darzustellen.

Beispiel:

```
        .
        .
        LD      HL,3039H     ;Binärwert laden
        CALL    0FAFH        ;in ASCII umwandeln und ausgeben
        .
        .
```

Auf dem Bildschirm wird der Wert 12345 (= 3039H) dargestellt.

CALL 132FH
Ganzzahl in ASCII umwandeln

Eine im Arbeitsbereich 1 befindliche Ganzzahl (Integer) wird in einen ASCII-String umgewandelt und bei der mit HL adressierten Stelle im Speicher abgelegt. Die Register B und C sollten beim Einsprung auf einen Wert größer 6 gesetzt werden, um eine Einfügung von Kommas oder Punkten im ASCII-String zu unterdrücken.

Das Ergebnis wird mit einem Endekennzeichen 00H versehen

Beispiel:

```
                .
                .
                LD      HL,456          ;umzuwandelnder Wert
                LD      (7921H),HL      ;in Arbeitsbereich 1
                LD      BC,606H         ;B u. C > 6 setzen
                LD      HL,STRING       ;Ergebnisfeld adressieren
                CALL    132FH           ;Wert in ASCII umwandeln
                .
                .
        STRING  DEFS    6               ;Ergebnisfeld
```

Nach der Umwandlung enthält das Ergebnisfeld den Eintrag
30H-30H-34H-35H-36H-00H = 00456.

CALL 0FBEH
Fließkomma-Wert in ASCII-String

Umwandlung einer Fließkommazahl einfacher oder doppelter Genauigkeit in einen ASCII-String. Die Fließkommazahl ist im Arbeitsbereich 1 bereitzustellen. Der erzeugte ASCII-String wird im Druckpuffer für formatierte Zahlenausgabe bei 7930H übergeben.

Der ASCII-String ist mit 00H abgeschlossen, HL zeigt beim Aussprung auf den Pufferanfang, DE auf das Ende des erzeugten ASCII-Strings (00H). Bei einem Feldüberlauf wird in das Byte vor dem Druckpuffer (792FH) das Zeichen '%' eingetragen.

Bei dieser Umwandlung kann eine komfortable Formatierung des zu erzeugenden ASCII-Strings angefordert werden. Diese Formatierung wird durch Einträge in den Registern A, B und C gesteuert.

Die einzelnen Bits des A-Registers haben folgende Auswirkungen auf die Formatierung:

Bit 7	= 0	- keine Formatierung durchführen.
	= 1	- Formatierung entsprechend den nachfolgend gesetzten Bits durchführen.
Bit 6	= 1	- alle 3 Stellen wird eine Komma zur Trennung der Tausender-Werte eingefügt.
Bit 5	= 1	- führende Leerzeichen des ASCII-Strings werden durch '*' ersetzt.
Bit 4	= 1	- vor der Zahl ist das Zeichen '$' auszugeben.
Bit 3	= 1	- Ein '+' als Vorzeichen ist darzustellen.
Bit 2	= 1	- Das Vorzeichen ist hinter der Zahl darzustellen.
Bit 1		- nicht benutzt
Bit 0	= 1	- ASCII-Darstellung mit Exponenten-Ausgabe

B-Register	=	Anzahl auszugebender Zeichen links des Dezimalpunktes.
C-Register	=	Anzahl auszugebender Zeichen rechts des Dezimalpunktes.

Beim Einsprung in 0FBDH anstelle von 0FBEH wird die Formatierung unterdrückt.

Beispiel:

```
                 .
                 .
        LD       HL,AWERT             ;Umwandlung eines ASCII-Strings
                 CALL   0E6CH         ;in einfache Genauigkeit
                 CALL   0FBDH         ;und zurück in einen ASCII-String
                 .
                 .
        AWERT    DEFM   '1234.56'     ;Ausgangswert
                 DEFB   0             ;Endekennung
```

Der Ausgangs-ASCII-String '1234.56' wird zunächst in eine Zahl einfacher Genauigkeit umgewandelt.
Mit dem CALL 0FBDH erfolgt die Rückwandlung in den Original-ASCII-String, der nun im Bereich 7930H..... mit einem führenden Leerzeichen und einem abschließenden 00H steht.
Register HL enthält den Eintrag 7930H und Register DE den Eintrag 7938H.
Das Ergebnisfeld hat hexadezimal folgenden Inhalt:
20H-31H-32H-33H-34H-2EH-35H-36H-00H

Arithmetische Routinen

Diese Routinen führen arithmetische Operationen zwischen zwei Operanden desselben Datentyps aus.

Die Routinen erwarten die Operanden in den vorgegebenen Registern oder Arbeitsbereichen. Das Typ-Flag sollte vor dem Aufruf auf den entsprechenden Datentyp gesetzt werden.

Die Divisions-Routinen benutzen das Disvisions-Unterprogramm im Kommunikationsbereich (7880H-788DH); dieses muß dort unversehrt vorhanden sein.

Routinen zur Bearbeitung von Ganzzahlen (Integer)

Die folgenden 5 Routinen führen arithmetische Operationen zwischen zwei 16-Bit Ganzzahlen aus. Die beiden Operanden sind in den Registerpaaren HL und DE bereitzustellen. Bis auf eine Ausnahme bleibt der Inhalt von DE unverändert erhalten; das Ergebnis wird in aller Regel im Registerpaar HL zurückgegeben.

CALL ØBD2H

Zwei Ganzzahlen addieren

Addiert den Inhalt des Registerpaares DE auf den Inhalt des Registerpaares HL. Die Summe wird in HL übergeben.

übersteigt die Summe jedoch 2^{15} (überlauf), so werden beide Werte zunächst in ein Fließkommazahl einfacher Genauigkeit umgewandelt und die Operation wiederholt. In diesem Fall steht das Ergebnis anschließend als Wert einfacher Genauigkeit im Arbeitsbereich 1 des Kommunikationsbereichs. Das Typ-Flag bei 78AFH erhält den Eintrag '4'.

Beispiel: Die in WERT1 und WERT2 gespeicherten Ganzzahlen sind zu addieren.

```
        .
        .
        LD      HL,WERT1     ;1. Wert laden
        LD      DE,WERT2     ;2. Wert laden
        LD      A,2          ;Typ-Flag = Ganzzahl
        LD      (78AFH),A
        CALL    ØBD2H        ;Werte addieren
        LD      A,(78AFH)    ;Typ-Flag laden
        CP      2            ;Ergebnis = Ganzzahl ?
        JR      NZ,SP        ;nein, ==> einf. Genauigkeit
INT     .                    ;ja!
        .
        .
WERT1   DEFW    15
WERT2   DEFW    4Ø
        .
```

CALL ØBC7H

Zwei Ganzzahlen subtrahieren

Subtrahiert den Wert in DE von dem Wert in HL. Die Differenz wird im Registerpaar HL übergeben.

Tritt ein ünterlauf auf, d.h. die Subtraktion von zwei Werten ungleichen Vorzeichens ergibt einen Wert > 2^{15}, so werden beide Werte vor der erneuten Subtraktion in Fließkommazahlen einfacher Genauigkeit umgewandelt. Die Differenz wird als Wert einfacher Genauigkeit im Arbeitsbereich 1 übergeben. Erkenntlich ist ein solcher Fall am Typ-Flag, das von '2' auf '4' gesetzt wird.

Beispiel: Von WERT1 soll WERT2 subtrahiert werden.

```
                .
                .
                LD      HL,WERT1      ;Werte laden
                LD      DE,WERT2
                LD      A,2           ;Typ=Ganzzahl
                LD      (78AFH),A
                CALL    0BC7H         ;DE von HL subtrahieren
                LD      A,(78AFH)     ;Typ-Flag laden
                CP      2             ;Ergebnis = Ganzzahl ?
                JR      NZ,SP         ;nein ==> einfache Genauigkeit
        INT     .                     ;Ja!
                .
        WERT1   DEFW    50
        WERT2   DEFW    30
```

CALL 0BF2H — Multiplikation von 2 Ganzzahlen

Der Inhalt von HL wird mit dem Inhalt von DE multipliziert. Das Produkt steht anschließend in HL.

Im Falle eines Überlaufs (Produkt > 2^{15}), werden beide Werte in Fließkommazahlen einfacher Genauigkeit umgewandelt und die Multiplikation erneut durchgeführt. Das Produkt steht in diesem Fall im Arbeitsbereich 1, das Typ-Flag enthält den Wert 4.

Beispiel: Der Inhalt von WERT1 ist mit WERT2 zu multiplizieren.

```
                .
                .
                LD      HL,WERT1      ;Werte laden
                LD      DE,WERT2
                LD      A,2           ;Typ-Flag = Ganzzahl
                LD      (78AFH),A
                CALL    0BF2H         ;Werte multiplizieren
                LD      A,(78AFH)     ;Typ-Flag laden
                CP      2             ;Ergebnis = Ganzzahl ?
                JR      NZ,SP         ;nein ==> einf. Genauigkeit
        INT     .                     ;ja!
                .
        WERT1   DEFW    18
        WERT2   DEFW    12
```

CALL 2490H

Division von Ganzzahlen

Der Inhalt von DE wird durch HL dividiert.

Beide Werte werden vor der Division in Fließkommazahlen einfacher Genauigkeit umgewandelt. Der Quotient wird ebenfalls mit einfacher Genauigkeit im Arbeitsbereich 1 übergeben. Das Typ-Flag bei 78AFH erhält den Eintrag '4'.

Die Inhalte von DE und HL werden zerstört.

Beispiel: WERT1 ist durch WERT2 zu dividieren.

```
        .
        .
        LD      DE,(WERT1)   ;Dividend laden
        LD      HL,(WERT2)   ;Divisor laden
        CALL    2490H        ;Division durchführen
        .
        .
WERT1   DEFW    80
WERT2   DEFW    4
```

CALL 0A39H

Vergleich von zwei Ganzzahlen

Die Inhalte von HL und DE werden algebraisch miteinander verglichen. Beide Registerinhalte bleiben unverändert.

Das Ergebnis des Vergleichs wird im A-Register und in den Status-Flags (Z = ZERO-Flag, C = CARRY-Flag) übergeben.

HL > DE	A = 1	
HL = DE	A = 0	Z-Flag = 1
HL < DE	A = -1	C-Flag = 1, S-Flag = 1

Beispiel: Die Inhalte von WERT1 und WERT2 sind zu vergleichen.

```
                .
                .
                LD      HL,(WERT1)   ;Werte laden
                LD      DE,(WERT2)
                CALL    0A39H        ;Werte vergleichen
                JP      Z,GLEICH     ;==> WERT1 = WERT2
                JP      C,KLEIN      ;==> WERT1 < WERT2
        GROSS   .                    ;==> WERT1 > WERT2
                .
        WERT1   DEFW    .....
        WERT2   DEFW    .....
```

Arithmetische Operationen einfacher Genauigkeit

Fünf weitere Routinen stehen zur Verfügung, Fließkommazahlen einfacher Genauigkeit arithmetisch miteinander zu verknüpfen.

Diese Routinen erwarten ein Argument in den Registerpaaren BC und DE und das zweite Argument im Arbeitsbereich 1 des Kommunikationsbereichs. Das Ergebnis wird grundsätzlich im Arbeitsbereich 1 zur Verfügung gestellt.

CALL 0716H Addition einfacher Genmauigkeit

Addieren von zwei Fließkommazahlen einfacher Genauigkeit.

Ein Summand ist in BC/DE bereitzustellen, der zweite Summand im Arbeitsbereich 1. Nach der Addition steht die Summe im Arbeitsbereich 1.

Beispiel:

```
                .
                LD      HL,WERT1     ;1. Wert adressieren
                CALL    9B1H         ;in Arbeitsbereich 1 übertragen
                LD      HL,WERT2     ;2. Wert adressieren
                CALL    9C2H         ;in BC/DE übertragen
                CALL    716H         ;beide Werte addieren
                .
                .
        WERT1   DEFS    4            ;Wert einfacher Genauigkeit
        WERT2   DEFS    4            ;Wert einfacher Genauigkeit
```

Das Ergebnis der Addition steht im Arbeitsbereich 1.

CALL 0713H

Subtraktion einfacher Genauigkeit

Subtrahiert eine in BC/DE stehende Fließkommazahl einfacher Genauigkeit vom Inhalt des Arbeitsbereichs 1. Die Differenz steht anschließend im Arbeitsbereich 1.

Beispiel:

```
                .
                .
                LD      HL,WERT1      ;1. Wert adressieren
                CALL    9B1H          ;in Arbeitsbereich 1 übertragen
                LD      HL,WERT2      ;2. Wert adressieren
                CALL    9C2H          ;in BC/DE übertragen
                CALL    716H          ;WERT1 - WERT2
                .
                .
        WERT1   DEFS    4             ;Wert einfacher Genauigkeit
        WERT2   DEFS    4             ;Wert einfacher Genauigkeit
```

CALL 0847H

Multiplikation einfacher Genauigkeit

Multipliziert den im Arbeitsbereich 1 stehenden Wert einfacher Genauigkeit mit dem Inhalt von BC/DE. Das Produkt steht anschließend im Arbeitsbereich 1.

Beispiel:

```
                .
                .
                LD      HL,WERT1      ;1. Wert adressieren
                CALL    9B1H          ;in Arbeitsbereich 1 übertragen
                LD      HL,WERT2      ;2. Wert adressieren
                CALL    9C2H          ;in BC/DE übertragen
                CALL    847H          ;WERT1 * WERT2
                .
                .
        WERT1   DEFS    4             ;Wert einfacher Genauigkeit
        WERT2   DEFS    4             ;Wert einfacher Genauigkeit
```

CALL 2490H

Division einfacher Genauigkeit

Dividiert eine in BC/DE stehende Fließkommazahl einfacher Genauigkeit durch den Inhalt des Arbeitsbereichs 1 (einfache Genauigkeit). Der Quotient steht anschließend im Arbeitsbereich 1.

Beispiel:

```
                .
                .
                LD      HL,WERT1      ;Divisor adressieren
                CALL    9B1H          ;in Arbeitsbereich 1 übertragen
                LD      HL,WERT2      ;Dividend adressieren
                CALL    9C2H          ;in BC/DE
                CALL    2490H         ;WERT2 / WERT1
                .
                .
        WERT1   DEFS    4             ;Wert einfacher Genauigkeit
        WERT2   DEFS    4             ;Wert einfacher Genauigkeit
```

CALL 0A0CH

Vergleich von Werten einfacher Genauigkeit

Diese Routine führt einen algebraischen Vergleich von zwei Fließkommazahlen durch. Diese müssen in den Registern BC/DE und im Arbeitsbereich 1 bereitgestellt werden.

Das Ergebnis des Vergleichs wird im A-Register und im Flag-Register übergeben.

BC/DE	>	Arb.ber.1	-	A = - 1	Carry- + Sign-Flags = 1
BC/DE	<	Arb.ber.1	-	A = 1	
BC/DE	=	Arb.ber.1	-	A = 0	Zero-Flag = 1

Beispiel:

```
                .
                .
                LD      HL,WERT1      ;1. zu vergleichenden Wert
                CALL    9B1H          ;in Arbeitsbereich 1
                LD      HL,WERT2      ;2. zu vergleichenden Wert
                CALL    9C2H          ;in BC/DE
                CALL    0A0CH         ;Werte vergleichen
                JP      Z,...         ;==> WERT1 = WERT2
                JP      C,...         ;==> WERT1 < WERT2
                .                     ;==> WERT1 > WERT2
                .
        WERT1   DEFS    4             ;Wert einfacher Genauigkeit
        WERT2   DEFS    4             ;Wert einfacher Genauigkeit
```

Arithmetische Operationen doppelter Genauigkeit

Wie für die beiden vorherigen Datentypen stehen auch für die arithmetische Verknüpfung von zwei Fließkommazahlen doppelter Genauigkeit fünf Routinen zur Verfügung.

Dabei ist ein Argument im Arbeitsbereich 1 (791DH-7924H) und das zweite Argument im Arbeitsbereich 2 (7927H-792EH) bereitzustellen. Das Ergebnis steht immer im Arbeitsbereich 1.

CALL 0C77H — Addition doppelter Genauigkeit

Hier werden zwei Fließkommazahlen doppelter Genauigkeit addiert und die Summe im Arbeitsbereich 1 übergeben.

Beispiel:

```
                .
                .
                LD      A,8          ;Typ-Flag = doppelte Genauigkeit
                LD      (78AFH),A    ;setzen
                LD      DE,WERT1     ;1. Argument adressieren
                LD      HL,791DH     ;Arbeitsbereich 1 adressieren
                CALL    9D3H         ;1. Argument in Arbeitsbereich 1
                LD      DE,WERT2     ;2. Argument adressieren
                LD      HL,7927H     ;Arbeitsbereich 2 adressieren
                CALL    9D3H         ;2. Argument in Arbeitsbereich 2
                CALL    0C77H        ;WERT1 + WERT2
                .
                .
        WERT1   DEFS    8            ;Wert doppelter Genauigkeit
        WERT2   DEFS    8            ;Wert doppelter Genauigkeit
```

CALL 0C70H — Subtraktion doppelter Genauigkeit

Subtrahiert eine im Arbeitsbereich 2 befindliche Fließkommazahl doppelter Genauigkeit vom Inhalt des Arbeitsbereichs 1. Die Differenz steht anschließend im Arbeitsbereich 1.

Beispiel:

```
                .
                .
                LD      A,8         ;Typ-Flag = doppelte Genauigkeit
                LD      (78AFH),A   ;setzen
                LD      DE,WERT1    ;1. Wert adressieren
                LD      HL,791DH    ;Arbeitsbereich 1 adressieren
                CALL    9D3H        ;1. Wert in Arbeitsbereich 1
                LD      DE,WERT2    ;2. Wert adressieren
                LD      HL,7927H    ;Arbeitsbereich 2 adressieren
                CALL    9D3H        ;2. Wert in Arbeitsbereich 2
                CALL    0C70H       ;WERT1 - WERT2
                .
                .
        WERT1   DEFS    8           ;Wert doppelter Genauigkeit
        WERT2   DEFS    8           ;Wert doppelter Genauigkeit
```

CALL 0DA1H — Multiplikation doppelter Genauigkeit

Multipliziert die Fließkommazahl doppelter Genauigkeit im Arbeitsbereich 2 mit dem Inhalt von Arbeitsbereich 1. Das Produkt steht anschließend im Arbeitsbereich 1.

Beispiel:

```
                .
                .
                LD      A,8         ;Typ-Flag = doppelte Genauigkeit
                LD      (78AFH),A   ;setzen
                LD      DE,WERT1    ;1. Wert adressieren
                LD      HL,791DH    ;Arbeitsbereich 1 adressieren
                CALL    9D3H        ;1. Wert in Arbeitsbereich 1
                LD      DE,WERT2    ;2. Wert adressieren
                LD      HL,7927H    ;Arbeitsbereich 2 adressieren
                CALL    9D3H        ;2. Wert in Arbeitsbereich 2
                CALL    0DA1H       ;WERT1 * WERT2
                .
                .
        WERT1   DEFS    8           ;Wert doppelter Genauigkeit
        WERT2   DEFS    8           ;Wert doppelter Genauigkeit
```

CALL ØDE5H

Division doppelter Genauigkeit

Dividiert eine im Arbeitsbereich 1 befindliche Fließkommazahl doppelter Genauigkeit durch eine solche im Arbeitsbereich 2. Der Quotient steht anschliessend im Arbeitsbereich 1.

Beispiel:

```
                .
                .
                LD      A,8          ;Typ-Flag = doppelte Genauigkeit
                LD      (78AFH),A    ;setzen
                LD      DE,WERT1     ;Dividend adressieren
                LD      HL,791DH     ;Arbeitsbereich 1 adressieren
                CALL    9D3H         ;Dividend in Arbeitsbereich 1
                LD      DE,WERT2     ;Divisor adressieren
                LD      HL,7927H     ;Arbeitsbereich 2 adressieren
                CALL    9D3H         ;Divisor in Arbeitsbereich 2
                CALL    ØDE5H        ;WERT1 / WERT2
                .
                .
        WERT1   DEFS    8            ;Wert doppelter Genauigkeit
        WERT2   DEFS    8            ;Wert doppelter Genauigkeit
```

CALL ØA4FH

Vegleich doppelter Genauigkeit

Vergleicht zwei Fließkommazahlen doppelter Genauigkeit miteinander. Diese sind in den Arbeitsbereichen 1 und 2 bereitzustellen. Das Ergebnis des Vergleichs wird im A-Register und im Flag-Register dargestellt.

```
        Arb.Ber.1 > Arb.Ber.2   -   A = 1
        Arb.Ber.1 = Arb.Ber.2   -   A = Ø    Zero-Flag = 1
        Arb.Ber.1 < Arb.Ber.2   -   A = -1   Carry- + Sign-Flag = 1
```

Beispiel:

```
                .
                .
                LD      A,8          ;Typ-Flag = doppelte Genauigkeit
                LD      (78AFH),A    ;setzen
                LD      DE,WERT1     ;1. Wert adressieren
                LD      HL,791DH     ;Arbeitsbereich 1 adressieren
                CALL    9D3H         ;Wert 1 in Arbeitsbereich 1
                LD      DE,WERT2     ;2. Wert adressieren
                LD      HL,7927H     ;Arbeitsbereich 2 adressieren
                CALL    9D3H         ;Wert 2 in Arbeitsbereich 2
                CALL    ØA4FH        ;Wert 1 mit Wert 2 vergleichen
```

```
        JP      Z,...           ;==> WERT1 = WERT2
        JP      C,...           ;==> WERT1 < WERT2
        .                       ;==> WERT1 > WERT2
        .
WERT1   DEFS    8               ;Wert doppelter Genauigkeit
WERT2   DEFS    8               ;Wert doppelter Genauigkeit
```

Mathematische Routinen

Die nachfolgenden Routinen dienen der Berechnung mathematischer Funktionen. Diese erhalten beim Aufruf, bis auf eine Ausnahme, nur ein Argument, das im Arbeitsbereich 1 des Kommunikationsbereichs zu übergeben ist. Der Typ des Arguments ist im Typ-Flag bei 78AFH anzugeben.

CALL 0977H — Absolut-Wert ermitteln ABS(N)

Der im Arbeitsbereich 1 befindliche Wert wird in sein positives Äquivalent umgewandelt. Das Ergebnis steht anschließend ebenfalls im Arbeitsbereich 1.

Steht im Arbeitsbereich 1 die negative Ganzzahl -32768, so wird das Ergebnis als Fließkommazahl einfacher Genauigkeit übergeben. Das Typ-Flag bei 78AFH wird entsprechend korrigiert.

Als Argument sind alle Datentypen zugelassen.

Beispiel: Von der im Feld WERT1 befindlichen Fließkommazahl einfacher Genauigkeit soll der Absolutwert ermittelt werden.

```
        .
        .
        LD      A,4             ;Typ-Flag = einfache Genauigkeit
        LD      (78AFH),A       ;setzen
        LD      HL,WERT1        ;Argument adressieren
        CALL    9B1H            ;und in Arbeitsbereich 1 übertr.
        CALL    0977H           ;Absolutwert bilden
        .
        .
```

```
WERT1     DEFB   0F2H          ;Wert einfacher Genauigkeit
          DEFB   80H           ;= - 94.3456
          DEFB   0BCH
          DEFB   87H
```

Nach Abschluß der Operation steht im Arbeitsbereich 1 der Wert 94.3456 in einfacher Genauigkeit.

CALL 0B37H

Ermitteln der nächstniedrigeren ganzen Zahl INT (N)

Diese Routine ermittelt den ganzzahligen Anteil einer Fließkommazahl. Diese ist im Arbeitsbereich 1 bereitzustellen, das Typ-Flag muß den korrekten Datentyp anzeigen.

Wenn die Wertgröße es erlaubt (-32768 bis +32767), wird das Ergebnis im Datentyp 'Ganzzahl' zurückgegeben, ansonsten bleibt der Datentyp unverändert. Der Typ des Ergebnisses kann im Typ-Flag bei 78AFH ermittelt werden.

Das Ergebnis steht im Arbeitsbereich 1.

Beispiel: Von der in Wert 1 befindlichen Fließkommazahl einfacher Genauigkeit ist der ganzzahlige Anteil zu ermitteln.

```
          .
          .
          LD     A,4           ;Typ-Flag = einfache Genauigkeit
          LD     (78AFH),A     ;setzen
          LD     HL,WERT1      ;Argument adressieren
          CALL   9B1H          ;in Arbeitsbereich 1 übertragen
          CALL   0B37H         ;ganzzahligen Anteil ermitteln
          .
          .
WERT      DEFB   00H           ;Wert einfacher Genauigkeit
          DEFB   6FH           ;= 12.5896
          DEFB   49H
          DEFB   84H
```

Das Ergebnis, die Zahl 12 steht als Datentyp 'Ganzzahl' im Arbeitsbereich 1, das Typ-Flag bei 78AFH hat den Eintrag '2'.

CALL 15BDH

Arkustangens ermitteln ATN (N)

Von einem im Arbeitsbereich 1 als Fließkommazahl gespeicherten Tangens-Wert wird der dazugehörige Winkel im Bogenmaß ermittelt. Das Ergebnis wird als Fließkommazahl im Arbeitsbereich zur Verfügung gestellt.

Beispiel: Von dem im Feld 'TAN' gespeicherten Tangens-Wert ist der Winkel im Bogenmaß zu ermitteln und ins Feld 'RAD' zu übertragen.

```
          .
          .
          LD      A,4          ;Typ-Flag = einfache Genauigkeit
          LD      (78AFH),A    ;setzen
          LD      HL,TAN       ;Tangens-Wert
          CALL    09B1H        ;in Arbeitsbereich 1 übertragen
          CALL    15BDH        ;Winkel ermitteln
          LD      HL,RAD       ;Ergebnisfeld adressieren
          LD      DE,7921H     ;Arbeitsbereich 1 adressieren
          CALL    9D3H         ;Ergebnis ins Ergebnisfeld
          .
          .
TAN       DEFB    3AH          ;Tangens von 30° (0.57735)
          DEFB    0CDH
          DEFB    13H
          DEFB    80H
          .
RAD       DEFS    4            ;Ergebnisfeld
```

Nach Ablauf der o.a. Routine enthält das Feld 'RAD' den Wert des Winkels 30° im Bogenmaß (91H-0AH-06H-80H = 0.523598)

CALL 1541H

Kosinus eines Winkels ermitteln COS (N)

Ermittelt den Kosinus eines im Bogenmaß angegebenen Winkels.

Der Winkel ist als Fließkommazahl im Arbeitsbereich 1 bereitzustellen, das Ergebnis wird ebenfalls als Fließkommazahl im Arbeitsbereich 1 übergeben.

Beispiel: Der Kosinus des im Feld 'RAD' angegebenen Winkels ist zu ermitteln und in das Feld 'KOS' zu übertragen.

```
            .
            .
            LD      A,4         ;Typ-Flag = einfache Genauigkeit
            LD      (78AFH),A   ;setzen
            LD      HL,RAD      ;Winkel in Arbeitsbereich 1
            CALL    9B1H        ;übertragen
            CALL    1541H       ;Kosinus ermitteln
            LD      HL,KOS      ;Ergebnisfeld adressieren
            LD      DE,7921H    ;Arbeitsbereich 1 adressieren
            CALL    9D3H        ;Ergebnis übernehmen
            .
            .
RAD         DEFB    91H         ;Bogenmaß von 30°
            DEFB    0AH         ;(0.523598)
            DEFB    06H
            DEFB    80H
            .
KOS         DEFS    4           ;Ergebnisfeld
```

Nach der Berechnung steht im Feld 'KOS' der Kosinus des Winkels von 30°
(D7H-B3H-5DH-80H = 0.866025)

CALL 1547H

Sinus eines Winkels ermitteln SIN (N)

Es wird der Sinus eines Winkels ermittelt.

Der Winkel ist im Bogenmaß als Fließkommazahl im Arbeitsbereich 1 bereitzustellen. Das Ergebnis steht anschließend ebenfalls im Arbeitsbereich 1.

Beispiel: Von dem im Feld 'RAD' befindlichen Winkel ist der Sinus zu ermitteln und im Feld 'SIN' zu übergeben.

```
            .
            .
            LD      A,4         ;Typ-Flag = einfache Genauigkeit
            LD      (78AFH),A   ;setzen
            LD      HL,RAD      ;Winkel in Arbeitsbereich 1
            CALL    9B1H        ;übertragen
            CALL    1547H       ;Sinus ermitteln
```

```
         LD      HL,SIN       ;Ergebnisfeld adressieren
         LD      DE,7921H     ;Arbeitsbereich 1 adressieren
         CALL    9D3H         ;Ergebnis übertragen
         .
         .
RAD      DEFB    91H          ;30° im Bogenmaß (= 0.523598)
         DEFB    0AH
         DEFB    06H
         DEFB    80H
         .
SIN      DEFS    4            ;Ergebnisfeld
```

Das Feld 'SIN' enthält nach der Berechnung den Sinus von 30°.

CALL 1439H — Ermitteln der Exponentialfunktion e^X EXP (N)

Ermittelt den Wert e zur Basis N (e = 2.71828).

Das Argument N ist als Fließkommazahl einfacher Genauigkeit im Arbeitsbereich 1 bereitzustellen. Das Ergebnis wird ebenfalls in einfacher Genauigkeit im Arbeitsbereich 1 übergeben.

Beispiel: $e^{1.5708}$ ist zu ermitteln.

```
         .
         .
         LD      A,4          ;Typ-Flag = einfache Genauigkeit
         LD      (78AFH),A    ;setzen
         LD      HL,EXP       ;Exponenten adressieren
         CALL    9B1H         ;in Arbeitsbereich 1 übertragen
         CALL    1439H        ;Funktion berechnen
         LD      DE,7921H     ;Arbeitsbereich 1 adressieren
         LD      HL,ERG       ;Ergebnisfeld adressieren
         CALL    9D3H         ;Ergebnis übernehmen
         .
         .
EXP      DEFB    0DBH         ;Exponent (1.5708)
         DEFB    0FH
         DEFB    49H
         DEFB    81H
         .
ERG      DEFS    4            ;Ergebnisfeld
```

JP 13F2H Potenzieren X^Y

Die Ausgangswerte X und Y sind als Fließkommazahlen einfacher Genauigkeit bereitzustellen.

Die Basis 'X' ist in den Stack-Bereich zu übertragen, der Exponent 'Y' ist im Arbeitsbereich 1 bereitzustellen. Das Ergebnis steht nach der Berechnung als Fließkommazahl einfacher Genauigkeit im Arbeitsbereich 1.

Hierbei ist eine Besonderheit zu beachten. Da eines der Argumente auf dem Stack bereitgestellt werden muß, kann die Routine nicht mit CALL aufgerufen werden, da dann die Rücksprungadresse als letzter Eintrag auf dem Stack stehen würde. Die Rücksprungadresse muß vielmehr vor dem Argument auf den Stack geschrieben und die Routine mit einem JP aufgerufen werden. Das nachfolgende Beispiel veranschaulicht dieses Verfahren.

Beispiel: 16^4 soll berechnet werden.

```
        .
        .
        LD      HL,RET          ;Rücksprungadresse
        PUSH    HL              ;auf den Stack schreiben
        LD      A,4             ;Typ-Flag = einfache Genauigkeit
        LD      (78AFH),A       ;setzen
        LD      HL,BAS          ;Basiswert adressieren
        CALL    9B1H            ;in Arbeitsbereich 1 übertragen
        CALL    9A4H            ;Arbeitsbereich 1 auf den Stack
        LD      HL,EXP          ;Exponenten adressieren
        CALL    9B1H            ;in Arbeitsbereich 1 übertragen
        JP      13F2H           ;x^y ermitteln
RET     .                       ;Rücksprungadresse
        .
        .
BAS     DEFW    0               ;Basiswert = 16
        DEFW    85H
        .
EXP     DEFW    0               ;Exponent = 4
        DEFW    83H
```

CALL 0809H

Natürlicher Logarithmus LOG (N)

Ermittelt den natürlichen Logarithmus einer Fließkommazahl einfacher Genauigkeit.
Das Argument ist im Arbeitsbereich 1 bereitzustellen, das Ergebnis wird ebenfalls als Fließkommazahl einfacher Genauigkeit im Arbeitsbereich 1 übergeben.

Beispiel: Der natürliche Logarithmus von 5 ist zu ermitteln.

```
                .
                .
                LD      A,4         ;Typ-Flag = einfache Genauigkeit
                LD      (78AFH),A   ;setzen
                LD      HL,ARG      ;Argument adressieren
                CALL    9B1H        ;in Arbeitsbereich 1 übertragen
                CALL    0809H       ;nat. Logarithmus ermitteln
                LD      DE,7921H    ;Arbeitsbereich 1 adressieren
                LD      HL,LOG      ;Ergebnisfeld adressieren
                CALL    9D3H        ;Ergebnis übernehmen
                .
                .
        ARG     DEFW    0           ;Argument = 5
                DEFB    02H
                DEFB    83H
                .
        LOG     DEFS    4           ;Ergebnisfeld
```

CALL 13E7H

Wurzel von N ermitteln SQR (N)

Ermittelt die Wurzel eines im Arbeitsbereich 1 gespeicherten Wertes.

Das Ergebnis wird ebenfalls im Arbeitsbereich 1 zurückgegeben.

Beispiel: Die Wurzel aus 144 ist zu ermitteln

```
                .
                .
                LD      A,4         ;Typ-Flag = einfache Genauigkeit
                LD      (78AFH),A   ;setzen
                LD      HL,ARG      ;Argument adressieren
                CALL    9B1H        ;in Arbeitsbereich 1 übertragen
                CALL    13E7H       ;Wurzel ziehen
                .
```

```
ARG     DEFW   0          ;Argument = 144
        DEFB   10H
        DEFB   88H
```

Nach der Berechnung steht das Ergebnis (= 12) im Arbeitsbereich 1.

CALL 14C9H

Zufallszahl ermitteln RND (N)

Erzeugt eine Zufallszahl zwischen 0 und 1 oder zwischen 1 und N, abhängig vom Wert 'N', der im Arbeitsbereich 1 übergeben werden muß.

Die erzeugte Zufallszahl wird als Fließkommazahl einfacher Genauigkeit im Arbeitsbereich 1 zurückgegeben und das Typ-Flag entsprechend modifiziert.

Das übergebene Argumnent 'N' bestimmt den Bereich der Zufallszahl. Ist N = 0, so wird eine Zufallszahl zwischen 0 und 1 erzeugt. Ist N > 0, so wird eine Zufallszahl zwischen 1 und N ermittelt und als ganze Zahl übergeben.

Beispiel: Eine Zufallszahl zwischen 1 und 20 soll ermittelt werden.

```
.
.
LD      A,2          ;Typ-Flag = Ganzzahl setzen
LD      (78AFH),A
LD      HL,20        ;N = 20 laden
LD      (7921H),HL   ;und in Arbeitsbereich 1
CALL    14C9H        ;Zufallszahl ermitteln
.
.
```

Nach der Berechnung steht eine Zufallszahl zwischen 1 und 20 mit einfacher Genauigkeit im Arbeitsbereich 1.

Die so erzeugte Zufallszahl ist keine echte Zufallszahl, sondern wird nach einem festen Algorithmus aus der letzten Zufallszahl gebildet.

Soll mehr Zufälligkeit hineingebracht werden, so kann mit einem CALL 1D3H ein neuer Basiswert gesetzt werden, der dem augenblicklichen Stand des Z80 Refresh-Registers entnommen wird. Der Basiswert und die letzte erzeugte Zufallszahl befinden sich bei 78AAH - 78ADH im Kommunikationsbereich.

RESTART - Vektoren

Im unteren Adreßbereich des Z80 befinden sich in 8er Schritten sogenannte Restart-Adressen, die mit einem speziellen 'RST' - Befehl erreichbar sind.

Die RESTART-Vektoren 8H bis 30H sind auf RAM-Erweiterungsausgänge am Anfang des Kommunikationsbereichs herausgeführt. Von dort erfolgt für die Vektoren 8H - 20H ein Sprung in ROM-Routinen, die auch für einen Maschinensprache- oder Assembler-Programmierer nützliche Funktionen erfüllen.

Es ist auch möglich, die RAM-Erweiterungsausgänge mit Sprungbefehlen in eigene Routinen zu belegen. Es ist jedoch dabei zu bedenken, daß nicht mehr so ohne weiteres alle anderen der in diesem Kapitel aufgeführten Routinen nutzbar sind, da diese gelegentlich auch Gebrauch von den Restart-Befehlen machen.

RST 8 — Ein Zeichen prüfen

Diese Routine dient dem BASIC-Interpreter zur Syntax-Prüfung einer Eingabezeile. Es wird das durch HL adressierte Zeichen mit dem auf den RST 8 - Befehl folgenden Zeichen verglichen. Sind beide gleich, wird automatisch die RST 10 - Routine aufgerufen und von dort auf das 2. Byte nach dem RST 8 - Befehl zurückgesprungen. Das HL-Registerpaar steht auf dem nächsten durch RST 10 ermittelten gültigen Zeichen, das A-Register enthält dieses Zeichen.

Stimmen die beiden Zeichen nicht überein, wird eine Fehlermeldung "SYNTAX ERROR" erzeugt und zur Eingabe-Phase des BASIC-Interpreters zurückgesprungen.

Beispiel: Überprüfung eines durch HL adressierten Textes, ob dieser aus den Zeichen 'A=B' besteht.

```
.
LD      HL,TEXT       Text-String adressieren
RST     8             ;A überprüfen
DEFB    'A'
RST     8             ;= überprüfen
DEFB    '='
RST     8             ;B überprüfen
DEFB    'B'
.
```

Stimmt eines der Zeichen nicht überein, wird ein SYNTAX ERROR erzeugt.

RST 10H

Nächstes gültiges Zeichen ermitteln

Diese Routine wird vom BASIC-Interpreter benutzt, um bei Analyse einer Eingabezeile das nächste gültige Zeichen zu ermitteln und zu übernehmen.

Das Registerpaar HL wird zur Adressierung eines Textstrings verwendet. Beim Einsprung wird die darin enthaltene Adresse um 1 erhöht. Das nächste von HL adressierte Zeichen wird ins A-Register geladen und das Carry-Flag entsprechend der Zeichenart gesetzt.

CARRY = 0 - Zeichen nicht numerisch
CARRY = 1 - Zeichen numerisch

Das ZERO-Flag wird gesetzt, wenn das Textende erreicht wurde. Dieses ist durch ein Byte = 00H zu kennzeichnen.

Bei der Bearbeitung des Textstrings bleiben Leerzeichen und die Steuerzeichen 09H (TAB) und 0AH (LF) unberücksichtigt, sie werden einfach übergangen.

Beispiel: HL zeigt auf eine Programmzeile, die eine Wertzuweisung enthält. Die Zeile wurde bis zum Gleichheitszeichen bearbeitet. Es ist zu prüfen, ob darauf eine Variable oder eine Konstante folgt.

```
        .
        .
NEXT    RST     10H         ;nächstes Zeichen laden
        JR      NC,NONUM    ;nicht numerisch!
        CALL    1E5AH       ;Konstante laden
        JP      .....       ;weiter
NONUM   CP      '+'         ;ist es ein '+'-Vorzeichen?
        JR      Z,NEXT      ;ja, nächstes Zeichen laden
        CP      '-'         ;ist es ein '-'-Vorzeichen?
        JR      Z,NEXT      ;ja, nächstes Zeichen
        CALL    260DH       ;Variable in Variablentabelle
                            ;ermitteln
        .
        .
```

RST 18H

DE mit HL vergleichen

Diese Routine führt einen logischen Vergleich der beiden Registerpaare DE und HL durch. Sie arbeitet nicht korrekt bei Ganzzahlen mit Vorzeichen, dazu ist die Routine bei 0A39H (arithmetischer Vergleich HL:DE) zu verwenden.

Das Ergebnis wird in den ZERO- und CARRY-Flags angezeigt. Der Inhalt des A-Registers wird verändert.

```
CARRY  =  1    -    HL < DE
CARRY  =  0    -    HL >= DE
ZERO   =  1    -    gleich
ZERO   =  0    -    ungleich
```

Beispiel: Es ist zu prüfen, ob ein in DE befindlicher Wert im Bereich 200 - 500 liegt.

```
        .
        .
        LD      HL,500      ;oberen Grenzwert laden
        RST     18H         ;mit Wert in DE vergleichen
        JR      C,FEHLER    ;Wert in DE > 500
        LD      HL,199      ;unteren Grenzwert laden
        RST     18H         ;mit Wert in DE vergleichen
        JR      NC,FEHLER   ;Wert in DE < 200
        .
        .
```

RST 20H

Datentyp ermitteln

Das Typ-Flag bei 78AFH wird ausgewertet und im A-Register ein numerischer Wert in Abhängigkeit zum dort angezeigten Datentyp zurückgegeben. Das Flag-Register kann ebenfalls ausgewertet werden.

Typ	Status	A-Register
Ganzzahl	NZ,C,M,E	-1
String	Z,C,P,E	0
einf.Genauigk.	NZ,C,P,O	1
dopp.Genauigk.	NZ,NC,P,E	5

Der im A-Register übergebene Wert entspricht dem Typ-Code in 78AFH - 3.

Die Routine wird vom BASIC-Interpreter dazu benutzt, den Datentyp eines im Arbeitsbereich 1 gespeicherten Wertes zu ermitteln. Aber Vorsicht, Typ-Flag und Arbeitsbereich 1 müssen nicht immer synchron sein.

Beispiel: Nach der Addition zweier Ganzzahlen ist zu prüfen, ob das Ergebnis als Ganzzahl oder als Fließkommazahl einfacher Genauigkeit übergeben wurde.

```
              .
              .
              LD      A,2           ;Typ-Flag = Ganzzahl
              LD      (78AFH),A     ;setzen
              LD      DE,WERT1      ;1. Summanden laden
              LD      HL,WERT2      ;2. Summanden laden
              CALL    0BD2H         ;Addition zweier Ganzzahlen
              RST     20H           ;Typ des Ergebnisses testen
              JP      M,INT         ;==> Ergebnis = Ganzzahl
SP            .                     ;==> Ergebnis = einf. Genauigkeit
              .
              .
INT           .
              .
WERT1         DEFW    2             ;1. Summand als Ganzzahl
WERT2         DEFW    2             ;2. Summand als Ganzzahl
```

Übertragungs - Routinen

In diesem Abschnitt werden einige Routinen beschrieben, die Daten verschiedener Typen innerhalb des Speichers oder zwischen Registern und Speicher übertragen.

CALL 09B4H

Fließkommazahl einfacher Genauigkeit von BC/DE in den Arbeitsbereich 1

Überträgt eine Fließkommazahl einfacher Genauigkeit aus den Registerpaaren BC/DE in den Arbeitsbereich 1 des Kommunikationsbereichs.

Der Inhalt von HL wird zerstört, BC/DE bleiben unverändert.

Achtung: Das Typ-Flag bei 78AFH wird nicht aktualisiert.

Beispiel: Zwei im Speicher befindliche Fließkommazahlen einfacher Genauigkeit sind zu addieren.

```
        .
        .
        LD      DE,ZAHL1    ;MSB + Exponent der 1. Zahl laden
        LD      BC,ZAHL1+2  ;LSB und NSB der 1. Zahl laden
        CALL    09B4H       ;in den Arbeitsbereich 1 übertr.
        LD      DE,ZAHL2    ;MSB + Exponent der 2. Zahl laden
        LD      BC,ZAHL2+2  ;LSB und NSB der 2. Zahl laden
        CALL    0716H       ;auf Arbeitsbereich 1 addieren
        .
        .
ZAHL1   DEFS    4           ;Wert einfacher Genauigkeit
ZAHL2   DEFS    4           ;Wert einfacher Genauigkeit
```

ZAHL1 und ZAHL2 müssen die Werte in der Folge LSB-NSB-MSB-EXP enthalten.

CALL 09B1H

Fließkommazahl einfacher Genauigkeit in Arbeitsbereich 1 übertragen

Eine im Speicher befindliche Fließkommazahl einfacher Genauigkeit wird in den Arbeitsbereich 1 übertragen. HL muß die Anfangsadresse des Speicherbereichs enthalten.

Die Inhalte von HL/BC/DE werden zerstört.

Beispiel:

```
        .
        .
        LD      HL,WERT     ;Wert im Speicher adressieren
        CALL    9B1H        ;in Arbeitsbereich 1 übertragen
        .
        .
WERT    DEFS    4           ;enthält eine Fließkommazahl
                            ;einfacher Genauigkeit
```

CALL Ø9CBH

Fließkommazahl einfacher Genauigkeit vom Arbeitsbereich 1 übernehmen

Eine im Arbeitsbereich 1 befindliche Fließkommazahl wird in den Programmspeicher übernommen. HL muß die Anfangsadresse des Speicherbereichs enthalten.

Die Inhalte aller Register werden verändert.

Beispiel:

```
              .
              .
              LD      HL,WERT     ;Speicheradresse laden
              CALL    9CBH        ;Wert aus Arbeitsbereich 1
              .                   ;in Speicher übertragen
              .
      WERT    DEFS    4           ;enthält nach dem CALL die
                                  ;Fließkommazahl einfacher Genau-
                                  ;igkeit.
```

CALL Ø9C2H

Fließkommazahl einfacher Genauigkeit aus dem Speicher in die Register BC/DE

Lädt eine Fließkommazahl einfacher Genauigkeit aus einem Speicherbereich in die Registerpaare BC und DE.

HL muß die Anfangsadresse des Speicherbereichs enthalten.
Die Inhalte aller Register werden verändert.

Beispiel: Zwei Zahlen einfacher Genauigkeit sind zu addieren. Das Ergebnis ist in BC/DE zu übertragen.

```
              .
              .
              LD      HL,WERT1    ;1. Summanden adressieren
              CALL    9B1H        ;in Arbeitsbereich 1
              LD      HL,WERT2    ;2. Summanden adressieren
              CALL    9C2H        ;in BC/DE
              CALL    716H        ;BC/DE auf Arbeitsbereich 1 add.
              LD      BC,(7923H)  ;Exponent und MSB der Summe
              LD      DE,(7921H)  ;NSB und LSB der Summe laden
              .
              .
      WERT1   DEFS    4           ;Wert einfacher Genauigkeit
      WERT2   DEFS    4           ;Wert einfacher Genauigkeit
```

WERT1 und WERT2 enthalten die Summanden in der Form
LSB-NSB-MSB-EXP.

CALL Ø9BFH

Fließkommazahl einfacher Genauigkeit vom Arbeitsbereich 1 in BC/DE

überträgt eine Fließkommazahl einfacher Genauigkeit aus dem Arbeitsbereich 1 in die Registerpaare BC/DE.

Achtung, es wird nicht geprüft, ob der Arbeitsbereich 1 wirklich eine Fließkommazahl einfacher Genauigkeit enthält, dies muß vom aufrufenden Programm sichergestellt werden.

Beispiel: Zwei Fließkommazahlen einfacher Genauigkeit sind zu multiplizieren. Das Ergebnis ist in BC/DE bereitzustellen.

```
        .
        .
        LD      HL,WERT1      ;Multiplikant adressieren
        CALL    9B1H          ;in Arbeitsbereich 1 übertragen
        LD      HL,WERT2      ;Multiplikator adressieren
        CALL    9C2H          ;in BC/DE übertragen
        CALL    847H          ;Multiplikation ausführen
        CALL    9BFH          ;Produkt in BC/DE übertragen
        .
        .
WERT1   DEFS    4             ;Multiplikant in einf. Genauigk.
WERT2   DEFS    4             ;Multiplikator in einf. Genauigk.
```

Die Einträge in WERT1 und WERT2 müssen in der Form
LSB-NSB-MSB-EXP vorhanden sein.

CALL Ø9A4H

Arbeitsbereich 1 auf den Stack übertragen

übeträgt eine Fließkommazahl einfacher Genauigkeit aus dem Arbeitsbereich auf den Stack. Diese wird dort in der Reihenfolge LSB-NSB-MSB-EXP gespeichert.

Alle Registerinhalte bleiben unverändert.

Es wird dabei nicht geprüft, ob sich im Arbeitsbereich 1 wirklich eine Fließkommazahl einfacher Genauigkeit befindet.

Diese Routine wird zum Beispiel benötigt, wenn Sie die Potenzier-Routine bei 13F2H benutzen wollen. Dort ist die Basis als Fließkommazahl einfacher Genauigkeit auf dem Stack bereitzustellen.

CALL 09D3H

Variable übertragungs - Routine

überträgt, abhängig vom Datentyp, in der Länge des Typ-Flags (78AFH) einen Wert von der in DE spezifizierten Adresse zu der in HL spezifizierten Adresse.

Die Register A, DE und HL werden verändert.

Beispiel: Eine Variable doppelter Genauigkeit mit dem Namen 'XY' soll in der Variablentabelle ermittelt und der Wert der Variablen ins Programm übernommen werden.

```
            .
            .
            LD      HL,NAME     ;Variablennamen adressieren
            CALL    260DH       ;Variablenadresse ermitteln
                                ;DE enthält die Variablenadresse
            RST     20H         ;ist es doppelte Genauigkeit?
            JR      NC,OK       ;ja, alles klar
            JP      ERROR       ;nein, Fehler
OK          LD      HL,DP       ;Speicherbereich adressieren
            CALL    9D3H        ;Variable übertragen
            .
            .
NAME        DEFM    'XY'        ;Name der Variablen
            DEFB    0           ;Endekennung
DP          DEFS    8           ;Empfangsfeld für die Variable
```

CALL 29C8H

übertragen einer Stringvariablen

Eine String-Variable wird aus dem String-Bereich oder der Programmtabelle in einen programminternen Speicherbereich übertragen.

Bei Einsprung muß HL die Adresse der String-Variablen in der Variablen-Tabelle enthalten und DE die Empfangsadresse im Programm.

Für eine String-Variable hat ein Eintrag in der Variablen-Tabelle folgendes Format:

    1. Byte   =   Stringlänge
    2.+3.Byte   =   Stringadresse

Beispiel: Eine Stringvariable mit dem Namen 'A$' soll in ein programminternes Feld 'VAR' übertragen werden.

```
              .
              .
              LD     HL,NAME       ;Variablennamen adressieren
              CALL   260DH         ;in Variablen-Tabelle ermitteln
              RST    20H           ;Ist es eine String-Variable?
              JR     Z,OK          ;ja, alles klar
              JP     FEHLER        ;nein, Fehler
    OK        EX     DE,HL         ;Adresse der Variablen-Tab. in HL
              LD     DE,VAR        ;Empfangsfeld adressieren
              CALL   29CBH         ;Variable übertragen
              .
              .
    NAME      DEFM   'A$'          ;Variablen-Name
              DEFB   0             ;Endekennung
    VAR       DEFS   255           ;Empfangsfeld
```

BASIC - Funktionen

BASIC-Funktionen unterscheiden sich von den vorausgegangenen Funktionen hauptsächlich in der intensiven Benutzung des Kommunikationsbereichs.

Bei Anwendung der nachfolgenden Routinen ist dafür Sorge zu tragen, daß der Kommunikationsbereich intakt ist und nicht durch das aufrufende Maschinenprogramm zerstört oder anderweitig genutzt wurde.

Die Routinen eignen sich besonders zum Einsatz in Maschinenprogramm-Unterroutinen, die aus einem BASIC-Programm heraus aufgerufen werden.

CALL 1B2CH

Zeile im Programm ermitteln

Diese Routine durchsucht die Programm-Tabelle nach einer BASIC-Zeile mit einer vorgegebenen Zeilennummer. Die zu ermittelnde Zeilennummer ist im Registerpaar DE bereitzustellen.

Alle Register werden verändert. Beim Rücksprung kann der Erfolg der Aktion an den Status-Flags abgelesen werden. BC und HL enthalten entsprechende Adressinformationen.

Status	Flags	Registerinhalte
Zeile gefunden	C/Z	BC = Startadresse der Zeile in der Programmtabelle
Zeile nicht gefunden, Zeilennummer zu groß	NC/Z	HL/BC = Endadresse des Programms + 1
Zeile nicht gefunden, jedoch größere Zeilennummern im Programm.	NC/NZ	BC = Adresse der Zeile mit der nächsthöheren Zeilennummer HL = Adresse der darauf folgenden Zeile im Programm.

Beispiel: Die Programmzeile mit der Zeilennummer 500 soll im Programm ermittelt werden. Wenn vorhanden, ist im A-Register der Wert 0, wenn nicht vorhanden, der Wert -1 zu übergeben.

```
        .
        .
        LD      DE,500      ;Zeilennummer laden
        CALL    1B2CH       ;im Programm suchen
        LD      A,1         ;Kennung für nicht gefunden
        JR      NC,A1       ;Zeile nicht vorhanden
        XOR     A           ;A=0 für gefunden
A1      .
        .
```

Die Registerpaare HL und BC enthalten entsprechend dem Rücksprungstatus die erforderlichen Adreßangaben gemäß o.a. Tabelle.

CALL 260DH

Adresse einer Variablen ermitteln

Mit dieser Routine kann die Variablen-Tabelle nach einer bestimmten Variablen durchsucht werden. Der Name der gesuchten Variablen ist in einem programminternen Feld bereitzustellen und mit HL zu adressieren.

Ist die Variable nicht vorhanden, so wird ein neuer Eintrag in der Variablen-Tabelle vorgenommen. Das Wertfeld wird = 0 gesetzt.

Beim Rücksprung enthält das Registerpaar DE die Adresse des ersten Werteintrags, das Typ-Flag bei 78AFH zeigt den Typ der gefundenen Variablen an.

Einfache Variable oder Elemente einer Matrix können mit dieser Routine ermittelt werden. Bei einer Matrix ist der Index wie bei einem BASIC-Zugriff dem Namen anzufügen, z.B. 'A(20)' für das 20. Element der Matrix 'A'.

Beispiel: Die Adresse der Variablen 'AB' soll ermittelt werden.

```
          .
          .
          LD      HL,NAME      ;Variablen-Name adressieren
          CALL    260DH        ;Variablen-Adresse ermitteln
          LD      (ADR),DE     ;Adresse übernehmen
          .
          .
NAME      DEFM    'AB'         ;Variablen-Name
          DEFB    0            ;Endekennung
ADR       DEFW    0            ;nach dem Aufruf
                               ;Adresse der Variablen 'AB'
```

CALL 1EB1H

GOSUB - Emulation

Diese Routine erlaubt den Aufruf einer BASIC-Unterroutine aus einem Maschinenprogramm. Nach Ausführung der BASIC-Routine wird das Programm mit dem auf den CALL folgenden Befehl fortgesetzt.

Alle Register werden verändert.

Beim Einsprung muß HL die Anfangsadresse eines ASCII-Strings enthalten, in dem die erste Zeilennummer der BASIC-Unterroutine angegeben ist.

Beispiel: Eine BASIC-Unterroutine, beginnend bei Zeilennummer 800, soll aus einem Maschinenprogramm aufgerufen werden.

```
               .
               .
               LD     HL,ZEILE     ;Zeilennummer adressieren
               CALL   1EB1H        ;BASIC-Routine aufrufen
               .
               .
        ZEILE  DEFM   '800'        ;1. Zeilennumer der BASIC-Routine
               DEFB   0
```

Dies ist nur ein kleiner Ausschnitt der verfügbaren Routinen des LASER-ROM. Durch intensives Studium der dokumentierten ROM-Auflistung können eine Vielzahl weiterer Routinen lokalisiert und für einen Aufruf aus Maschinenprogrammen nutzbar gemacht werden.

aktiv computern

Die Home-Computer Laser 110, 210, 310 und VZ 200 verdanken ihre Popularität nicht zuletzt dem komfortablen und umfangreichen BASIC-Interpreter, der sich in Speicherbausteinen (ROMs) im Innern des Rechners befindet und dem Anwender nach dem Einschalten in all seinen Funktionen vollständig zur Verfügung steht.
Ziel dieses Buches ist es, die wesentlichen Funktionen des BASIC-ROMs zu beschreiben, damit Sie die internen Vorgänge in Ihrem Rechner besser verstehen und alle Funktionen optimal nutzen können. Das Buch soll auch dem Assembler-/Maschinenprogramm-Experten die Möglichkeit eröffnen, Funktionen des BASIC-ROMs in eigenen Programmen zu nutzen, sei es, um einfache Datenkonvertierungen auszuführen, die Ein-/Ausgabeschnittstellen zu benutzen oder mathematische Funktionen nicht selber programmieren zu müssen.

VOGEL-BUCHVERLAG
WÜRZBURG

ISBN 3-8023-0874-3